helion.pl

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Java 2. Podstawy

Autorzy: Cay S. Horstmann, Gary Cornell
Tłumaczenie: Maciej Gorywoda
ISBN: 83-7197-984-3
Tytuł oryginału: Core Java 2 Volume 1 Fundamentals
Format: B5, stron: 740

„Java 2. Podstawy" jest najlepszą książką dla programistów. Nie ma sobie równych pod względem ilości przekazanych informacji, a mimo to pozostaje bardzo czytelna. — K. N. King, Computer Reviews

Programiści znajdą tu wszystko, czego potrzeba, aby wykorzystać potęgę języka Java... — PC Week

„Java 2. Podstawy" dostarcza wszystkiego, czego potrzebujesz, aby ukończyć nawet najbardziej skomplikowane projekty. Dlatego właśnie ta książka przez pięć lat była międzynarodowym bestsellerem. „Java 2. Podstawy" omawia podstawy platformy Java 2 Standard Edition w wersji 1.3, zawiera pełny opis zorientowanego obiektowo środowiska programistycznego Javy, komponentów interfejsu użytkownika Swing i wiele, wiele więcej.

Piąta wydanie prezentuje wiele nowych, gotowych do zastosowania programów, z których zasłynęły poprzednie edycje tej książki: pojawiły się również nowe programy dotyczące mechanizmu odbicia oraz optymalizacji kodu. Szczegółowo wyjaśniono koncepcję klas wewnętrznych, funkcjonowanie proxy, obsługę wyjątków, techniki usuwania błędów, model zdarzeń Javy, interfejs wejścia/wyjścia oraz zarządzania plikami.

Kompendium dla programistów Javy, a w nim:

- Omówienie języka Java i tworzenia aplikacji w środowisku Forte(tm)
- Tworzenie aplikacji GUI przy pomocy nowych klas Swing
- Sposoby pisania klas proxy i klas wewnętrznych
- Korzystanie z modelu zdarzeń Javy
- Omówienie klas strumieni oraz mechanizmu zarządzania plikami

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Podziękowania 11

Przedmowa 13

Do Czytelnika 13
O książce 15

Rozdział 1. Wprowadzenie do Javy 17

Java jako narzędzie programistyczne 18
Zalety Javy 19
Koncepcje języka Java 20
Prosty 20
Zorientowany obiektowo 21
Rozproszony 22
Niezawodny 22
Bezpieczny 23
Neutralny pod względem architektury 24
Przenośny 24
Interpretowany 25
Wydajny 25
Wielowątkowy 25
Dynamiczny 26
Java i Internet 26
Aplety w działaniu 27
Java po stronie serwera 28
Krótka historia Javy 28
Główne nieporozumienia dotyczące Javy 31

Rozdział 2. Środowisko programistyczne Javy 35

Instalowanie pakietu Java Software Development Kit 36
Konfiguracja ścieżki dostępu 36
Instalowanie bibliotek i dokumentacji 37
Instalowanie przykładowych programów książki Core Java 38
Drzewo katalogów Javy 38
Różne środowiska programistyczne 39
Korzystanie z linii poleceń 40
Rozwiązywanie problemów 42
Praca ze zintegrowanym środowiskiem programistycznym 43
Wyszukiwanie błędów kompilacji 44
Kompilowanie i uruchamianie programów przy użyciu edytora tekstu 46
Aplikacje graficzne 48
Aplety 51

Rozdział 3. Podstawy programowania w Javie ... 57
Prosty program napisany w Javie ... 58
Komentarze ... 60
Typy danych ... 61
Liczby całkowite ... 62
Typy zmiennoprzecinkowe ... 62
Typ znakowy ... 63
Typ boolean ... 64
Zmienne ... 65
Przypisanie i inicjalizacja ... 66
Stałe ... 66
Operatory ... 67
Operatory inkrementacji i dekrementacji ... 69
Operatory relacji i typu boolean ... 69
Operatory bitowe ... 70
Funkcje i stałe matematyczne ... 71
Konwersje pomiędzy typami numerycznymi ... 72
Rzutowanie ... 73
Hierarchia nawiasów i operatorów ... 73
Łańcuchy ... 74
Konkatenacja ... 74
Wycinanie łańcuchów ... 75
Edycja łańcuchów ... 75
Porównywanie łańcuchów ... 77
Czytanie dokumentacji API online ... 79
Czytanie danych ... 81
Formatowanie wyjścia ... 83
Instrukcje sterujące ... 86
Blok instrukcji ... 86
Instrukcje warunkowe ... 87
Pętle nieokreślone ... 89
Pętle określone ... 93
Wielokrotny wybór — polecenie switch ... 96
Przerywanie instrukcji sterowania ... 98
Wielkie liczby ... 100
Tablice ... 102
Inicjalizacja tablic i tablice anonimowe ... 103
Kopiowanie tablic ... 104
Parametry linii poleceń ... 105
Sortowanie tablicy ... 106
Tablice wielowymiarowe ... 109
Tablice nierównej długości ... 112

Rozdział 4. Obiekty i klasy ... 115
Wprowadzenie do programowania zorientowanego obiektowo ... 116
Słownictwo OOP ... 117
Obiekty ... 118
Relacje pomiędzy klasami ... 119
Porównanie OOP z konwencjonalnymi technikami programowania proceduralnego ... 121
Korzystanie z istniejących klas ... 123
Obiekty i zmienne obiektów ... 123
Klasa biblioteczna GregorianCalendar ... 126

Tworzenie własnych klas ... 133
Klasa Pracownik ... 133
Używanie wielu plików źródłowych równocześnie ... 136
Analiza klasy Pracownik ... 137
Pierwsze doświadczenia z konstruktorami ... 137
Metody klasy Pracownik ... 139
Metody dostępu do danych prywatnych ... 142
Metody prywatne ... 143
Finalne pola składowe ... 143
Pola i metody statyczne ... 144
Pola statyczne ... 144
Stałe ... 145
Metody statyczne ... 146
Metody fabryczne ... 147
Metoda main ... 147
Parametry metod ... 150
Konstrukcja obiektów ... 155
Przeładowanie ... 155
Domyślna inicjalizacja pól składowych ... 156
Konstruktory domyślne ... 157
Bezpośrednia inicjalizacja pól składowych ... 157
Nazwy parametrów ... 158
Wywoływanie innego konstruktora ... 159
Bloki inicjalizacji ... 160
Niszczenie obiektów i metoda finalize ... 163
Pakiety ... 164
Wykorzystanie pakietów ... 164
Komentarze dokumentacji ... 172
Wstawianie komentarzy ... 172
Dokumentacja klasy ... 173
Dokumentacja metod ... 174
Dokumentacja pól składowych ... 174
Komentarze standardowe ... 174
Dokumentacja pakietu i podsumowanie ... 176
Generowanie dokumentacji ... 176
Porady dotyczące projektowania klas ... 177

Rozdział 5. Dziedziczenie ... 181

Rozszerzanie klas ... 181
Hierarchie dziedziczenia ... 188
Polimorfizm ... 189
Wiązanie dynamiczne ... 190
Zapobieganie dziedziczeniu — klasy i metody finalne ... 192
Rzutowanie ... 193
Klasy abstrakcyjne ... 195
Dostęp chroniony ... 200
Object — uniwersalna nadklasa ... 201
Metody equals i toString ... 202
Programowanie uniwersalne ... 209
Klasa ArrayList ... 211
Klasy opakowań ... 217
Klasa Class ... 221

Mechanizm odbicia ... 224
Wykorzystanie mechanizmu odbicia do analizy możliwości klas ... 225
Wykorzystanie mechanizmu odbicia do analizy obiektów w czasie działania programu ... 229
Wykorzystanie mechanizmu odbicia w pisaniu kodu uniwersalnej tablicy ... 235
Wskaźniki do metod ... 238
Zastosowanie dziedziczenia w projektowaniu ... 242

Rozdział 6. Interfejsy i klasy wewnętrzne ... 245

Interfejsy ... 246
Właściwości interfejsów ... 250
Interfejsy i klasy abstrakcyjne ... 251
Interfejsy i wywołania zwrotne ... 252
Klonowanie obiektów ... 255
Klasy wewnętrzne ... 260
Wykorzystanie klas wewnętrznych do kontaktowania się ze zmiennymi obiektu ... 262
Specjalne zasady składni dla klas wewnętrznych ... 266
Czy klasy wewnętrzne są użyteczne? Czy są w ogóle potrzebne? I czy są bezpieczne? ... 266
Lokalne klasy wewnętrzne ... 269
Statyczne klasy wewnętrzne ... 274
Proxy ... 277
Właściwości klas proxy ... 281

Rozdział 7. Programowanie grafiki ... 283

Wprowadzenie do zestawu Swing ... 284
Tworzenie ramek ... 287
Pozycjonowanie ramek ... 290
Wyświetlanie informacji w panelach ... 295
Figury 2D ... 299
Kolory ... 307
Wypełnianie figur geometrycznych ... 310
Tekst i czcionki ... 312
Obrazy ... 321

Rozdział 8. Obsługa zdarzeń ... 329

Podstawy obsługi zdarzeń ... 330
Przykład — obsługa kliknięcia przycisku ... 332
Wybór słuchaczy zdarzeń ... 338
Przykład — zmiana „wyglądu i wrażenia” ... 341
Przykład — przechwytywanie zdarzeń okna ... 344
Hierarchia zdarzeń AWT ... 347
Zdarzenia semantyczne i zdarzenia niskiego poziomu w AWT ... 350
Mechanizm obsługi zdarzeń — podsumowanie ... 350
Typy zdarzeń niskiego poziomu ... 353
Aktywacja komponentów ... 353
Zdarzenia dotyczące klawiatury ... 355
Pochłanianie zdarzeń ... 361
Zdarzenia dotyczące myszki ... 361
Działania ... 368
Multicasting ... 377
Kolejka zdarzeń ... 380
Dołączanie własnych zdarzeń ... 381

Rozdział 9. Komponenty Swing interfejsu użytkownika ... 391
Wzór projektu MVC ... 392
Analiza MVC przycisków zestawu Swing ... 396
Wprowadzenie do zarządzania układem graficznym ... 398
Układ krawędzi ... 400
Panele ... 401
Wprowadzanie tekstu ... 403
Pola tekstowe ... 403
Walidacja danych ... 409
Pola haseł ... 417
Obszary tekstowe ... 418
Etykiety i komponenty etykiet ... 421
Wybór tekstu ... 423
Edycja tekstu ... 424
Dokonywanie wyborów ... 426
Pola wyboru ... 426
Przełączniki ... 429
Krawędzie ... 433
Listy kombinowane ... 437
Suwaki ... 440
Menu ... 446
Tworzenie menu ... 446
Ikony elementów menu ... 449
Pola wyboru i przełączniki jako elementy menu ... 450
Menu kontekstowe ... 452
Mnemoniki i akceleratory ... 453
Włączanie i wyłączanie elementów menu ... 455
Paski narzędzi ... 459
Podpowiedzi ... 461
Zaawansowane techniki zarządzania układem graficznym ... 464
Układ siatkowy ... 467
Układ pudełkowy ... 471
Układ arkuszowy ... 476
Parametry gridx, gridy, gridweight oraz gridheight ... 478
Pola wagowe ... 478
Parametry fill i anchor ... 478
Poszycie ... 479
Alternatywna metoda określania parametrów gridx, gridy, gridwidth i gridheight ... 479
Praca bez menedżera układu ... 483
Tworzenie własnych menedżerów układu ... 483
Sekwencja dostępu ... 487
Okna dialogowe ... 489
Okna wyboru ... 490
Tworzenie okien dialogowych ... 501
Przesyłanie danych ... 505
Okno pliku ... 511
Okno wyboru koloru ... 522

Rozdział 10. Aplety ... 529
Podstawy apletów ... 529
Prosty aplet ... 532
Uruchamianie przeglądarki apletów ... 534

Oglądanie apletu w przeglądarce ... 535
Przekształcanie aplikacji w aplety ... 539
Cykl życia apletu ... 540
Podstawy systemu ochrony ... 542
Okna dialogowe w apletach ... 543
Znaczniki HTML oraz atrybuty apletów ... 545
Atrybuty określające położenie apletu ... 546
Atrybuty dotyczące kodu apletu ... 547
Atrybuty dotyczące przeglądarek nie połączonych z Javą ... 550
Znacznik OBJECT ... 550
Znaczniki Java Plug-In ... 551
Przekazywanie informacji apletom ... 552
Multimedia ... 557
Adresy URL ... 558
Tworzenie plików multimedialnych ... 559
Kontekst apletu ... 560
Komunikacja pomiędzy apletami ... 560
Wyświetlanie elementów w przeglądarce ... 561
Aplet „Spis adresów” ... 563
To aplet. Nie, to aplikacja. To jedno i to samo! ... 565
Pliki JAR ... 570
Manifest ... 572
Przechowywanie plików JAR ... 573
Autonomiczne pliki JAR ... 574
Zasoby ... 574
Pakiety opcjonalne ... 578
Pieczętowanie ... 579

Rozdział 11. Wyjątki i proces debugowania ... 581

Obsługa błędów ... 582
Klasyfikacja wyjątków ... 583
Powiadamianie o wyjątkach zwracanych przez metody ... 585
Jak wygenerować wyjątek? ... 587
Tworzenie klas wyjątków ... 588
Łapanie wyjątków ... 589
Wielokrotne łapanie wyjątków ... 591
Powtórne zwracanie tego samego wyjątku ... 592
Jeszcze jedno spojrzenie na obsługę błędów i wyjątków ... 595
Wskazówki dotyczące korzystania z wyjątków ... 599
Techniki debugowania ... 602
Metody debugowania ... 602
Stwierdzenia ... 606
Korzystanie z okna konsoli ... 609
Śledzenie zdarzeń AWT ... 610
Robot AWT ... 614
Profilowanie ... 618
Test wykorzystania ... 622
Korzystanie z debuggera ... 623
Debugger JDB ... 623
Debugger Forte ... 628

Rozdział 12. Strumienie i pliki ... 631
Strumienie ... 631
Wczytywanie i zapisywanie bajtów ... 632
Zoo pełne strumieni ... 634
Budowa filtrów strumieni ... 637
Strumienie danych ... 640
Strumienie plików swobodnego dostępu ... 643
Strumienie plików ZIP ... 653
Wykorzystanie strumieni ... 660
Zapisywanie separowanego tekstu ... 661
Klasa StringTokenizer oraz tekst separowany ... 662
Pobieranie separowanego tekstu ... 663
Strumienie swobodnego dostępu ... 666
Strumienie obiektów ... 673
Przechowywanie obiektów zmiennego typu ... 674
Format pliku serializacji obiektów ... 678
Problem zapisywania referencji obiektów ... 682
Format referencji obiektów ... 688
Ochrona ... 690
Wersje ... 695
Serializacja w roli klonowania ... 697
Zarządzanie plikami ... 700

Dodatek A Słowa kluczowe Javy ... 707

Skorowidz ... 709

10

Aplety

W tym rozdziale:

- podstawy apletów,
- znaczniki HTML oraz atrybuty apletów,
- multimedia,
- kontekst apletu,
- pliki JAR.

W poprzednich rozdziałach poznałeś już większość narzędzi Javy, znasz też podstawy programowania graficznego w tym języku. Mamy nadzieję, że podobnie jak my uważasz, iż Java jest dobrym (jeśli nie doskonałym) zorientowanym obiektowo językiem ogólnego przeznaczenia, a biblioteki interfejsu użytkownika Swing są elastyczne i użyteczne. To ważne zalety, ale to nie one są powodem całego tego zamieszania wokół Javy. Niesamowity zamęt, jaki powstał w czasie pierwszych kilku lat istnienia Javy (wspominaliśmy o tym w rozdziale 1.) wynikał z faktu, iż Java jest zdolna do „ożywienia" Internetu. Możesz tworzyć specjalne programy Javy (nazywane apletami), które mogą być pobierane z Internetu i uruchamiane przez przeglądarki internetowe połączone z Javą. W tym rozdziale nauczysz się pisać podstawowe aplety. Profesjonalizm apletów zależy od stopnia opanowania przez programistę mechanizmu wielowątkowości oraz programowania sieciowego w Javie. Są to bardziej skomplikowane tematy, które poruszymy w książce *Java 2. Techniki zaawansowane*.

Nowoczesne przeglądarki internetowe obsługują Dynamic HTML oraz języki skryptowe, co sprawia, że potrafią dużo więcej niż w chwili, gdy na rynku pojawiła się platforma Javy. Wciąż jednak aplety napisane w prawdziwym języku programowania mają większe możliwości niż jakakolwiek kombinacja języków HTML, XML i języków skryptowych.

Podstawy apletów

Do tej pory używałeś języka HTML (ang. *hypertext markup language* — hipertekstowy język znaczników) do opisywania układu graficznego stron internetowych. HTML to proste narzędzie służące do oznaczania elementów hipertekstu. Dla przykładu, `<TITLE>` oznacza tytuł

strony, a więc każdy tekst, który znajdzie się za tym znacznikiem, zostanie potraktowany jak tytuł danej strony. Koniec tytułu oznaczasz napisem </TITLE> (jest to jedna z podstawowych zasad znaczników — znak /, wraz z następującą po nim nazwą, oznacza koniec elementu).

Podstawowa zasada umieszczania apletów na stronach internetowych jest prosta — strona HTML musi poinformować przeglądarkę, jakie aplety powinny zostać załadowane, a następnie gdzie na stronie powinny się one znaleźć. Jak mogłeś tego oczekiwać, znacznik, którego musisz użyć, powinien informować przeglądarkę o tym:

- skąd pobierać pliki klas,
- jak wyświetlić aplet na stronie (rozmiar apletu, położenie itd.).

> Jednym ze sposobów umieszczania apletów na stronach internetowych jest użycie znacznika APPLET oraz parametrów przekazujących opisane powyżej informacje. Konsorcjum W3 zaproponowało zmianę znacznika na bardziej uniwersalną nazwę OBJECT, a znacznik APPLET został zanegowany przez specyfikację HTML 4.0. Najnowsze przeglądarki rozpoznają obydwa znaczniki, ale musisz pamiętać, że ich starsze wersje znają wyłącznie APPLET. W dalszej części tego rozdziału omówimy znaczniki HTML dotyczące apletów.

Następnie przeglądarka ściąga pliki klas z Internetu (lub z odpowiedniego katalogu na maszynie użytkownika) i uruchamia aplet przy użyciu wirtualnej maszyny Javy.

Oprócz apletu strona internetowa może zawierać wiele innych elementów HTML, z którymi na pewno już się spotkałeś: czcionki, listy, grafikę, linki i tak dalej. Aplety to tylko jedna z wielu koncepcji hipertekstu. Warto zdawać sobie sprawę z faktu, że język programowania Java *nie* jest narzędziem służącym do projektowania stron HTML — jest narzędziem *ożywiającym te strony*. Nie oznacza to, że elementy projektu GUI w aplecie Javy nie są ważne, ale muszą one współpracować i w gruncie rzeczy podlegać projektowi HTML, w którym są uruchamiane.

> W niniejszej książce nie omawiamy znaczników; zakładamy, że znasz — lub współpracujesz z kimś, kto zna — podstawy języka HTML. Do uruchomienia apletów Javy wymagana jest znajomość zaledwie kilku z nich. Oczywiście, w dalszej części tego rozdziału zapoznamy Cię z nimi. Z nauką języka HTML nie ma najmniejszego problemu — w najbliższej księgarni znajdziesz dziesiątki książek poświęconych temu językowi. Jedną z tych, które przekazują wszelkie potrzebne informacje, nie obrażając przy tym Twojej inteligencji, jest *HTML i XHTML. Przewodnik encyklopedyczny*, autorstwa Chucka Musciano i Billa Kennedy'ego, wydawnictwa Helion.

W czasie powstania apletów jedyną obsługującą je przeglądarką była HotJava firmy Sun. Oczywiście, niewielu użytkowników miało ochotę używać osobnej przeglądarki tylko po to, by móc cieszyć się nowymi zabawkami. Aplety Javy stały się naprawdę popularne, gdy Netscape dołączył obsługę Javy do swojej przeglądarki. Wkrótce to samo stało się z Internet Explorerem. Niestety, producenci przeglądarek internetowych pozostali daleko w tyle za Javą. Na szczęście i Netscape, i Internet Explorer obsługują Javę 1.0, a ich nowsze wersje radzą sobie z większością narzędzi Javy 1.1.

Mimo to musimy podkreślić, że obsługa apletów wiąże się z denerwującymi ograniczeniami i niezgodnością. Dla przykładu, Microsoft prawdopodobnie nigdy nie zaimplementuje tych elementów Javy, które uważa za konkurencyjne względem jego własnej technologii. Netscape

w zasadzie stracił już zainteresowanie maszynami wirtualnymi. Netscape 6 pozwala użytkownikom podłączyć do przeglądarki ich własną maszynę wirtualną, ale np. Netscape 4 nie posiada takiej możliwości. Wszystko to utrudnia tworzenie apletów korzystających z nowych właściwości Javy, a przy tym działających na różnych przeglądarkach.

Aby przezwyciężyć ten problem, Sun udostępnił narzędzie o nazwie „Java Plug-In" (początkowo nosiło ono nazwę „Activator"). Korzystając z mechanizmów rozszerzania aplikacji, obsługiwanych przez przeglądarki Internet Explorer i Netscape Navigator, Java Plug-In podłącza się do obydwu przeglądarek i umożliwia im uruchamianie apletów Javy przy użyciu zewnętrznego Java Runtime Environment (środowiska wykonawczego Javy), dostarczanego przez firmę Sun. Wszystko na to wskazuje, że JRE zawsze będzie używać najnowszej wersji maszyny wirtualnej, dzięki czemu zawsze będziesz mógł korzystać z najnowszych i najwspanialszych osiągnięć języka Java.

Gdy już zainstalujesz Java Plug-In, będziesz mógł wybrać odpowiadającą Ci wersję maszyny wirtualnej Javy. Aby uruchamiać aplety zamieszczone w tej książce, musisz zainstalować Java Plug-In i wybrać maszynę Java 1.3 (patrz rysunek 10.1).

Rysunek 10.1. *Wybór maszyny wirtualnej Javy w oknie Java Plug-In*

Oczywiście, jeżeli projektujesz stronę internetową dla szerokiego grona odbiorców, wymaganie, aby każdy użytkownik Twojej strony zainstalował Java Plug-In, w ogóle nie wchodzi w grę, ponieważ pobieranie go zajmuje trochę czasu. W takim wypadku musisz zaprojektować aplety, które będą działać z maszyną wirtualną Javy wbudowaną w Netscape Navigatora lub w Internet Explorera. Najlepiej jest trzymać się narzędzi Java 1.0 i tak bardzo uprościć aplet, jak to tylko możliwe. Jednak, szczerze mówiąc, jeżeli Twój aplet ma być tak prosty, to prawdopodobnie możesz się w ogóle obyć bez niego — w algorytmach możesz używać JavaScriptu, przekazywać dane za pomocą formularzy, a w animacjach używać animowanych plików GIF. Następnie cały program umieszczasz na serwerze, najlepiej na serwerze obsługującym oparte na Javie strony internetowe oraz serwlety.

Z drugiej strony, jeżeli piszesz bardzo skomplikowany program, zastanów się, czy wykorzystanie przeglądarki internetowej do uruchamiania go niesie ze sobą jakiekolwiek korzyści. Jeżeli nie, możesz po prostu udostępnić aplikację Javy, którą użytkownik będzie mógł ściągnąć i uruchomić na swoim komputerze. Wciąż będziesz korzystał ze wszystkich osiągnięć Javy, takich jak niezależność od platformy, zarządzanie ochroną danych oraz prosty dostęp do

sieci i baz danych. Oczywiście, korzystanie z przeglądarki także ma swoje zalety. Użytkownikowi zazwyczaj łatwiej jest zlokalizować aplikację na stronie internetowej niż w lokalnym systemie plików (aplikację łatwo zgubić, jeżeli nie korzystamy z niej na co dzień). Z kolei administratorowi łatwiej jest aktualizować aplikację na stronie internetowej niż wysyłać poprawki i udoskonalenia wszystkim zainteresowanym.

Programami Javy, które odniosły największy sukces, są aplikacje *intranetowe* pełniące funkcję interfejsów dla systemów informacyjnych. Dla przykładu, wiele firm zainwestowało w programy obliczające zwrot kosztów, wynajdujące potencjalnie korzystne transakcje, planujące harmonogramy i dni wolne od pracy, obsługujące wydawanie zleceń i tak dalej. Programy te są relatywnie małe, potrzebują kontaktu z bazami danych i elastyczności, jaką zapewniają formularze stron internetowych, oraz muszą być dostosowane do wymagań danej korporacji. Aplety są więc dla nich doskonałym narzędziem. A ponieważ liczba użytkowników jest ograniczona, nie ma problemu z dystrybucją Java Plug-In.

Zalecamy, abyś dla swoich aplikacji sieciowych wybrał jedno z następujących rozwiązań:

1. Jeżeli pracujesz w intranecie, używaj Java Plug-In do obsługi apletów Javy. Pozwoli Ci to na maksymalną kontrolę nad platformą Javy, zmniejszy ból głowy spowodowany przenośnością oraz da możliwość korzystania z najbardziej zaawansowanych narzędzi Javy. Oczywiście, w takim wypadku musisz zadbać o instalację Java Plug-In na komputerach użytkowników.

2. Jeżeli nie pracujesz w intranecie, nie używaj apletów. Przy obliczeniach korzystaj z języków skryptowych; przy animacji — z animowanych plików GIF; przy wpisywaniu danych — z formularzy oraz z przetwarzania danych po stronie serwera.

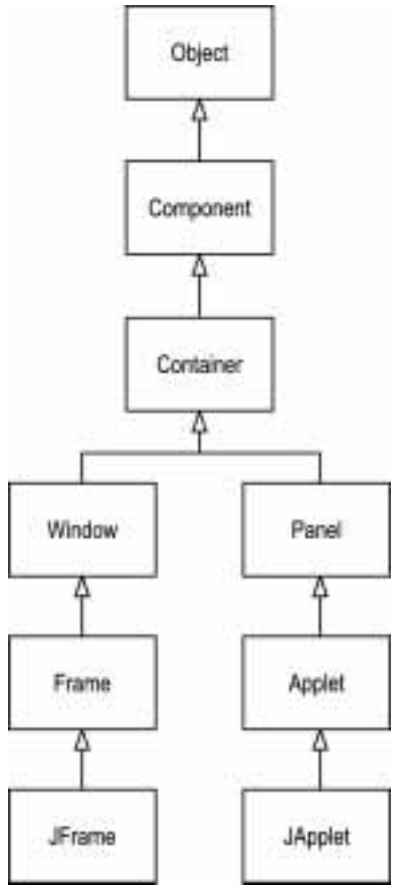
Prosty aplet

Aby zadośćuczynić tradycji, pierwszym apletem, jaki napiszemy, będzie `NieWitajSwiecie`. Zanim to zrobimy, musimy podkreślić, że z punktu widzenia programisty aplety nie są niczym nowym. Aplet to po prostu klasa Javy rozszerzająca (koniecznie) klasę `java.applet.Applet`. Zwróć uwagę, że pakiet `java.applet` nie jest częścią pakietu AWT, mimo iż aplet jest komponentem AWT, co widać na rysunku 10.2 przedstawiającym hierarchię dziedziczenia. W naszej książce do implementacji apletów będziemy używać zestawu Swing. Jak możesz się przekonać, spoglądając na rysunek 10.2, `JApplet` jest bezpośrednią podklasą klasy `Applet`.

> Jeżeli Twój aplet zawiera komponenty Swing, musisz rozszerzyć klasę `JApplet`. Komponenty Swing znajdujące się wewnątrz samej klasy `Applet` nie zostaną prawidłowo przerysowane.

Hierarchia dziedziczenia (rysunek 10.2) niesie ze sobą pewne oczywiste, lecz mimo to użyteczne konsekwencje. Na przykład, ponieważ aplety są komponentami AWT, obsługa zdarzeń w apletach działa dokładnie tak samo, jak omawiana w rozdziale 8. obsługa zdarzeń aplikacji.

Listing 10.1 zawiera kod apletu „Nie Witaj Świecie".



Rysunek 10.2. *Hierarchia dziedziczenia apletu*

Listing 10.1. *NieWitajSwiecieAplet.java*

```
 1. import java.awt.*;
 2. import javax.swing.*;
 3.
 4. public class NieWitajSwiecieAplet extends JApplet
 5. {
 6.    public void init()
 7.    {
 8.       Container powZawartosci = getContentPane();
 9.       JLabel etykieta = new JLabel("To nie jest aplet 'Witaj, świecie'",
10.          SwingConstants.CENTER);
11.       powZawartosci.add(etykieta);
12.    }
13. }
```

Zauważ, jak bardzo ten kod przypomina odpowiadający mu program z rozdziału 7. Jednakże, ponieważ aplet znajduje się wewnątrz strony internetowej, nie trzeba określać, w jaki sposób program kończy swoje działanie.

Uruchamianie przeglądarki apletów

Aby uruchomić aplet, wykonaj następujące czynności:

1. Skompiluj pliki źródłowe w pliki klas.
2. Utwórz plik HTML, który poinformuje przeglądarkę, który plik klasy ma załadować jako pierwszy oraz w jaki sposób ma wyświetlić aplet.

Zazwyczaj (ale niekoniecznie) nadajesz plikowi HTML taką samą nazwę, jaką nosi odpowiadająca mu nazwa pliku klasy. Tak więc, zgodnie z tradycją, nazwiemy nasz plik *NieWitajSwiecieAplet.html*. Oto zawartość tego pliku:

```
<APPLET CODE="NieWitajSwiecieAplet.class" WIDTH=300 HEIGHT=100>
</APPLET>
```

Przed uruchomieniem apletu w przeglądarce powinieneś przetestować go za pomocą programu *appletviewer*, który jest częścią Java SDK. Aby użyć przeglądarki apletów w naszym przykładowym programie, wpisz w linii poleceń

```
appletviewer NieWitajSwiecieAplet.html
```

Plikiem przetwarzanym przez przeglądarkę apletów jest plik HTML, a nie plik klasy. Rysunek 10.3 przedstawia przeglądarkę apletów wyświetlającą powyższy aplet.

Rysunek 10.3. *Aplet uruchamiany przez przeglądarkę apletów*

Możesz również uruchamiać aplety za pomocą swojego edytora lub środowiska programistycznego. W systemie Emacs wybierz z menu *JDE/Run Applet*. Używając programu Textpad, wybierz *Tools/Run Java Applet* lub użyj skrótu klawiaturowego *CTRL+3*. Zostanie wyświetlone okno dialogowe prezentujące wszystkie pliki HTML znajdujące się w aktualnym katalogu. Jeżeli naciśniesz klawisz *ESC*, Textpad automatycznie utworzy najmniejszy możliwy plik HTML. W systemie Forte po prostu załaduj stronę HTML zawierającą znacznik apletu. Forte obsługuje prostą przeglądarkę, prezentującą działanie apletu wewnątrz strony internetowej. Możesz również kliknąć prawym klawiszem myszki plik źródłowy i w zakładce *Execution* zmienić wartość *Executor* na *Applet Execution*.

Dzięki poniższej sztuczce możesz uniknąć tworzenia dodatkowych plików HTML. Dołącz znacznik apletu *do pliku źródłowego jako komentarz*:

```
/*
 <APPLET CODE="MojAplet.class" WIDTH=300 HEIGHT=300>
 </APPLET>
*/
public class MojAplet extends JApplet
. . .
```

Następnie uruchom przeglądarkę apletów, jako parametr w linii poleceń podając *plik źródłowy*:

```
appletviewer MojAplet.java
```

Nie polecamy tej procedury, ale może ona okazać się pomocna, jeśli zależy Ci na zminimalizowaniu liczby plików, o które musisz się martwić w czasie testowania programu.

Przeglądarka apletów przydaje się podczas wstępnych testów, ale w pewnym momencie będziesz zmuszony uruchomić swoje aplety w przeglądarce internetowej, aby zobaczyć je w ten sam sposób, w jaki będzie je oglądał użytkownik. Program przeglądarki apletów pokazuje wyłącznie aplet, pomijając otaczający go tekst HTML. Jeżeli plik HTML zawiera kilka apletów, przeglądarka apletów wyświetli po prostu kilka okienek.

Oglądanie apletu w przeglądarce

Jeżeli posiadasz przeglądarkę obsługującą Java 2, taką jak Netscape 6 lub Opera, wystarczy otworzyć w niej plik HTML. Jednakże aby uruchomić aplet w Internet Explorerze lub Netscape 4, musisz zainstalować w przeglądarce Java Plug-In i skonwertować plik HTML tak, aby wywoływał ten program.

Jeżeli zainstalowałeś Java 2 SDK, Java Plug-In został zainstalowany automatycznie. Jednakże jeżeli chcesz zainstalować Java Plug-In osobno, możesz go pobrać z witryny *http://java.sun.com/products/plugin*.

Gdy już zainstalujesz SDK lub Java Plug-In, wywołaj Panel sterowania Java Plug-In (patrz rysunek 10.4). Jeśli używasz Windows, przejdź do panelu sterowania Windows (*Start/Ustawienia/Panel sterowania*) i kliknij dwukrotnie ikonę *Java Plug-In*. Jeżeli używasz Solaris, wpisz polecenie

```
~/.netscape/java/ControlPanel
```

lub załaduj stronę

```
~/.netscape/java/ControlPanel.html
```

Następnie zaznacz pole *Show Java Console*. Dzięki temu za każdym razem, gdy uruchomisz Java Plug-In, pojawi się okno konsoli (patrz rysunek 10.5). Okno konsoli jest bardzo pomocne przy wyświetlaniu komunikatów o błędach.

Rysunek 10.4. *Panel sterowania Java Plug-In*

Rysunek 10.5.
Konsola Javy

Niestety, znaczniki HTML, których musisz używać, wywołując Java Plug-In, są dość nieporęczne. Zamiast pisać je własnoręcznie, łatwiej będzie Ci uruchomić konwerter HTML, dostarczany w tym celu przez firmę Sun (patrz rysunek 10.6). Możesz go ściągnąć z *http://java.sun.com/products/plugin1.3/converter.html*. Rozpakuj plik, np. do */usr/local/jdk/converter* lub *c:\jdk\converter*.

Rysunek 10.6.
Konwerter HTML Java Plug-In

Konwerter, na podstawie prostego pliku HTML zawierającego tradycyjny znacznik `APPLET`, tworzy skomplikowany kod HTML, wymagany przez różne przeglądarki do uruchomienia Java Plug-In.

Zanim rozpoczniesz, utwórz katalog *plugin* i skopiuj do niego swój plik lub pliki HTML.

Możesz uruchomić konwerter HTML w następujący sposób: przejdź do katalogu *plugin* i wydaj następujące polecenie:

```
java -classpath /usr/local/jdk/converter/classes HTMLConverter
```

lub

```
java -classpath c:\jdk\converter\classes HTMLConverter
```

Domyślnie konwerter wyświetla graficzny interfejs użytkownika (patrz rysunek 10.6). Możesz używać tego programu do konwersji wszystkich plików w danym katalogu. Jeżeli wywołasz program z katalogu *plugin*, wszystkie dane wejściowe zostaną wpisane automatycznie i będziesz mógł po prostu kliknąć przycisk *Convert*.

Możesz również skorzystać z języka skryptowego lub pliku wsadowego, umieszczając go w katalogu *jdk/converter/classes*, ale w takim wypadku musisz własnoręcznie określić katalog plików HTML.

Dla przykładu, w przypadku `NieWitajSwiecieAplet.html` zaczynamy od następującego prostego kodu HTML:

```
<APPLET CODE="MojAplet.class" WIDTH=300 HEIGHT=300>
</APPLET>
```

Wynik konwersji jest zawarty w listingu 10.2. Znaczniki tego kodu wyjaśnimy w dalszej części rozdziału.

Listing 10.2. *NieWitajSwiecieApletPlugin.html*

```
 1. <!--"CONVERTED_APPLET"-->
 2. <!-- CONVERTER VERSION 1.3 -->
 3. <OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
 4. WIDTH = 300 HEIGHT = 100  codebase="http://java.sun.com/products/plugin/1.3/
    ➥jinstall-13-win32.cab#Version=1,3,0,0">
 5. <PARAM NAME = CODE VALUE = "NieWitajSwiecieAplet.class" >
 6. <PARAM NAME="type" VALUE="application/x-java-applet;version=1.3">
 7. <PARAM NAME="scriptable" VALUE="false">
 8. <COMMENT>
 9. <EMBED type="application/x-java-applet;version=1.3"
     ➥CODE = "NieWitajSwiecieAplet.class" WIDTH = 300 HEIGHT = 100  scriptable=false
     ➥pluginspage="http://java.sun.com/products/plugin/1.3/
     ➥plugin-install.html"><NOEMBED></COMMENT>
10. </NOEMBED></EMBED>
11. </OBJECT>
12. <!--
13. <APPLET CODE = "NieWitajSwiecieAplet.class" WIDTH = 300 HEIGHT = 100>
14. </APPLET>
15. -->
16. <!--"END_CONVERTED_APPLET"-->
```

Jeśli wolisz korzystać z linii poleceń, możesz po prostu wpisać ścieżkę dostępu do pliku lub plików, które powinny zostać skonwertowane, np.:

```
java -classpath /usr/local/jdk/converter/classes HTMLConverter *.html
```

lub

```
java -classpath c:\jdk\converter\classes HTMLConverter MojAplet.html
```

Konwerter używający linii poleceń skonwertuje i utworzy pliki zapasowe w ten sam sposób, w jaki wyświetla okno dialogowe.

Domyślnie konwerter produkuje pliki HTML, które działają zarówno w przeglądarce Internet Explorer, jak i w Netscape Navigator. Istnieją jednak inne sposoby konwersji — jeśli chcesz dowiedzieć się więcej, zajrzyj do dokumentacji konwertera.

Konwerter zamienia zawartość wszystkich plików w danym katalogu, a stare wersje kopiuje do katalogu *katzrodlowy_BAK*. Dla przykładu, jeżeli umieściłeś oryginalne pliki w katalogu *plugin*, zostaną one zastąpione plikami skonwertowanymi, a oryginały zostaną skopiowane do *plugin_BAK*.

Nie podoba nam się ten rodzaj konwersji — nawet jeżeli konwerter zachowuje oryginalne pliki. Zazwyczaj wygodnie jest mieć pod ręką zarówno oryginalny, jak i skonwertowany plik. Zalecamy następującą procedurę:

1. Skopiuj pliki do katalogu tymczasowego.
2. Uruchom konwerter.
3. Zmień nazwy skonwertowanych plików na coś w rodzaju *MojApletPlugin.html*.
4. Przenieś pliki z powrotem do oryginalnego katalogu.

Gdy już skonwertujesz pliki, możesz uruchomić aplet w przeglądarce. Po prostu załaduj skonwertowany plik HTML do swojej przeglądarki internetowej. O ile Java Plug-In został prawidłowo zainstalowany, na ekranie pojawi się aplet (patrz rysunek 10.7). Jeżeli Java Plug-In nie został zainstalowany, Twoja przeglądarka powinna przeprowadzić Cię przez proces jego pobierania i instalacji.

Rysunek 10.7. *Oglądanie apletu w przeglądarce*

Testowanie apletów korzystających z Java Plug-In jest trochę bolesne, ponieważ musisz uruchamiać swoje pliki HTML przy użyciu konwertera. Jeżeli używasz Netscape 4, możesz zaoszczędzić sobie trochę pracy, pod warunkiem jednak, że Twoje aplety nie korzystają z żadnych narzędzi Java 2 oprócz zestawu Swing (wszystkie aplety w tym rozdziale spełniają ten warunek). Ściągnij z witryny *http://java.sun.com/products/jfc/#download-swing* dodatek do zestawu Swing. Umieść plik *swing.jar* w katalogu *Netscape\Communicator\Program\Java\Classes*. Teraz Netscape 4 będzie w stanie uruchamiać aplety korzystające z zestawu Swing — wystarczy, że załadujesz prosty plik HTML zawierający znacznik APPLET. Konwersja na znaczniki EMBED nie będzie już potrzebna.

W przypadku Internet Explorera procedura jest podobna, lecz bardziej skomplikowana. Szczegóły znajdziesz pod adresem *http://java.sun.com/products/jfc/tsc/articles/applets/index.html*.

Przekształcanie aplikacji w aplety

Konwersja aplikacji graficznej (czyli aplikacji korzystającej z AWT, którą możesz uruchomić przy użyciu interpretera `java`) na aplet, który możesz umieścić na stronie internetowej, jest dość prostym zadaniem. W gruncie rzeczy, cały kod interfejsu użytkownika pozostaje bez zmian.

Oto procedura, którą musisz wykonać, aby przekształcić aplikację w aplet:

1. Napisz stronę HTML zawierającą odpowiedni znacznik ładujący kod apletu.
2. Utwórz podklasę klasy `JApplet`. Nowa klasa musi być opatrzona słowem `public`. W przeciwnym wypadku aplet nie załaduje się.
3. Wytnij ze swojej aplikacji metodę `main`. Nie buduj ramki dla aplikacji. Twój program zostanie wyświetlony wewnątrz przeglądarki internetowej.
4. Przenieś kod inicjalizacyjny z konstruktora ramki do metody `init` apletu. Nie musisz własnoręcznie tworzyć obiektu apletu — przeglądarka sama utworzy instancję i uruchomi metodę `init`.
5. Usuń wywołanie `setSize`; rozmiar apletu jest określany przez parametry `WIDTH` i `HEIGHT` pliku HTML.
6. Usuń wywołanie `setDefaultCloseOperation`. Aplet nie może zostać zamknięty; kończy działanie, gdy użytkownik wyłącza przeglądarkę.
7. Jeżeli aplikacja wywołuje metodę `setTitle`, usuń to wywołanie. Aplety nie posiadają pasków tytułowych (możesz, oczywiście, nadać tytuł samej stronie internetowej, używając znacznika `TITLE`).
8. Nie wywołuj metody `show`. Aplety są wyświetlane automatycznie.

W celach ćwiczeniowych przekształcimy w aplet program kalkulatora z rozdziału 9. Na rysunku 10.8 możesz zobaczyć, jak wygląda kalkulator uruchamiany w przeglądarce.

Rysunek 10.8.
Aplet kalkulatora

Listing 10.3 zawiera stronę HTML. Zwróć uwagę, że oprócz znaczników apletu znajduje się w nim także pewien tekst.

Listing 10.3. *Kalkulator.html (przed skonwertowaniem go przy użyciu konwertera HTML)*

```
1. <HTML>
2. <TITLE>A Kalkulator</TITLE>
3. <BODY>
4. Oto kalkulator, na wypadek gdybyś nie mógł znaleźć własnego.
5. <APPLET CODE="ApletKalkulatora.class" WIDTH=180 HEIGHT=180>
6. </APPLET>
7. </BODY>
8. </HTML>
```

Listing 10.4 zawiera kod apletu. Zbudowaliśmy podklasę klasy `JApplet`, umieściliśmy kod inicjalizacyjny w metodzie `init` oraz usunęliśmy wywołania `setTitle`, `setSize`, `setDefaultCloseOperation` i `show`. Klasa `PanelKalkulatora` nie uległa żadnym zmianom, więc jej kod został pominięty.

Listing 10.4. *ApletKalkulatora.java*

```
 1. import java.awt.*;
 2. import javax.swing.*;
 3.
 4. public class ApletKalkulatora extends JApplet
 5. {
 6.    public void init()
 7.    {
 8.       Container powZawartosci = getContentPane();
 9.       PanelKalkulatora panel = new PanelKalkulatora();
10.       powZawartosci.add(panel);
11.    }
12. }
```

`java.applet.Applet`

- `void init()` wywoływana, gdy aplet jest wczytywany po raz pierwszy. Przeładuj tę metodę i umieść w niej kod inicjalizacyjny.
- `void setSize(int szerokosc, int wysokosc)` wymusza zmianę rozmiaru apletu. Byłaby to świetna metoda, gdyby tylko działała; niestety, obecne przeglądarki internetowe nie obsługują jej, ponieważ powoduje ona zmiany w układzie graficznym strony. Można jej natomiast używać w przeglądarce apletów i całkiem możliwe, że przyszłe przeglądarki będą ją obsługiwać, odświeżając stronę internetową za każdym razem, gdy zmieni się rozmiar apletu.

Cykl życia apletu

W klasie `Applet` znajdują się cztery metody, za pomocą których budujesz strukturę każdego poważnego apletu: `init`, `start`, `stop` oraz `destroy`. Poniżej znajduje się krótki opis każdej z tych metod.

- `init`

  Ta metoda jest używana do inicjalizacji danych wymaganych przez aplet. Działa mniej więcej jak konstruktor — system wywołuje ją, gdy Java uruchamia aplet po raz pierwszy. Zazwyczaj służy ona przetwarzaniu wartości przekazywanych przez znacznik `PARAM`, a także tworzeniu komponentów interfejsu użytkownika.

  Aplety mogą posiadać domyślny konstruktor, ale zazwyczaj cała inicjalizacja odbywa się w metodzie `init`.

- `start`

  Ta metoda jest wywoływana automatycznie *po* metodzie `init`. Jest również wywoływana za każdym razem, gdy użytkownik powróci do strony zawierającej dany aplet, jeżeli wcześniej opuścił ją, ładując inną stronę. Oznacza to, że metoda `start` może być uruchomiona wielokrotnie, w przeciwieństwie do metody `init`. Tak więc kod, który powinien być wykonywany tylko raz, zamieszczaj w metodzie `init`, nie w metodzie `start`. W metodzie `start` zazwyczaj znajduje się kod restartujący wątek apletu, czyli np. ponownie rozpoczynający animację. Jeżeli Twój aplet nie wykonuje działań, które powinny zostać zatrzymane, gdy użytkownik opuszcza stronę, nie musisz implementować tej metody (ani metody `stop`).

- `stop`

  Ta metoda jest wywoływana automatycznie, gdy użytkownik opuszcza stronę, na której znajduje się aplet. Dlatego też może być wielokrotnie wywołana dla tego samego apletu. Dzięki niej możesz zatrzymać czasochłonny algorytm, aby nie spowalniał pracy systemu operacyjnego, gdy użytkownik nie zwraca uwagi na aplet. Nie powinieneś wywoływać tej metody bezpośrednio. Jeżeli Twój aplet nie obsługuje animacji, plików dźwiękowych czy skomplikowanych obliczeń, metoda `stop` zazwyczaj nie jest do niczego potrzebna.

- `destroy`

  Java gwarantuje wywołanie tej metody, gdy przeglądarka internetowa zostanie wyłączona w standardowy sposób. Ponieważ środowiskiem apletów jest strona HTML, nie musisz zawracać sobie głowy usuwaniem panelu. Gdy przeglądarka zostanie wyłączona, Java dokona tego automatycznie. Tym, co *musisz* umieścić w metodzie `destroy`, jest kod zwalniający zasoby niezależne od pamięci, takie jak konteksty graficzne, których używał aplet. Oczywiście, jeżeli aplet jest aktywny, przed wywołaniem metody `destroy` zostanie wywołana metoda `stop`.

`java.applet.Applet`

- `void start()` przeładuj tę metodę, umieszczając w niej kod, który powinien zostać wykonany *za każdym razem*, gdy użytkownik odwiedzi stronę zawierającą aplet. Typowym zadaniem tej metody jest reaktywacja wątku.
- `void stop()` przeładuj tę metodę, umieszczając w niej kod, który powinien zostać wykonany *za każdym razem*, gdy użytkownik opuści stronę zawierającą aplet. Typowym zadaniem tej metody jest zawieszenie działania wątku.
- `void destroy()` przeładuj tę metodę, umieszczając w niej kod, który powinien zostać wykonany, gdy użytkownik wyłącza przeglądarkę. Typowym zadaniem tej metody jest wywołanie metody `destroy` dla obiektów systemowych.

Podstawy systemu ochrony

Ponieważ aplety zostały zaprojektowane tak, aby mogły być pobierane z odległych serwerów, a następnie wykonywane lokalnie, bardzo ważne są względy bezpieczeństwa. Jeżeli użytkownik włączy w swojej przeglądarce obsługę Javy, przeglądarka będzie ściągać kod apletów ze stron HTML i natychmiast go uruchamiać. Użytkownik nie jest w stanie potwierdzić lub zatrzymać operacji wykonywanych przez konkretny aplet. W związku z tym na aplety (w przeciwieństwie do aplikacji) zostały nałożone pewne ograniczenia. *Menedżer ochrony apletów* zwraca `SecurityException` (wyjątek ochrony) za każdym razem, gdy aplet próbuje naruszyć jedną z reguł dostępu (aby dowiedzieć się więcej o menedżerach ochrony, zajrzyj do *Java 2. Techniki zaawansowane*).

Jakie działania *mogą* podejmować aplety na wszystkich platformach? Mogą wyświetlać obrazy i wydawać sygnały dźwiękowe, pobierać naciśnięcia klawiszy i kliknięcia myszki oraz wysyłać wpisane przez użytkownika dane z powrotem na serwer. To wystarcza, aby prezentować informacje i rysunki, a także pobierać polecenia od użytkownika. Środowisko ograniczonego wykonywania apletów często nazywane jest „piaskownicą". Aplety „bawiące się w piaskownicy" nie mogą wpływać na system operacyjny użytkownika ani szpiegować go. W tym rozdziale będziemy zajmować się wyłącznie apletami działającymi w „piaskownicy".

W szczególności, aplety „bawiące się w piaskownicy":

- *nie mogą* uruchamiać programów na komputerze użytkownika;
- nie mogą komunikować się z serwerem innym niż ten, z którego zostały ściągnięte; serwer ten jest nazywany *serwerem inicjującym* (ang. *originating host*). Ta zasada jest często określana słowami „aplety mogą dzwonić wyłącznie do domu". Dzięki temu użytkownicy są chronieni przed apletami, które mogłyby szpiegować w zasobach intranetu;
- nie mogą sczytywać ani zapisywać plików na komputerze użytkownika;
- nie mogą zbierać informacji na temat komputera użytkownika, poza numerem wersji Javy zainstalowanej na tym komputerze, nazwą i wersją systemu operacyjnego oraz rodzaju znaków używanych przy rozdzielaniu plików (np. `\` lub `/`), ścieżek dostępu (np. `:` lub `;`) i linii (np. `\n` lub `\r\n`). Aplety nie mogą poznać nazwiska użytkownika, jego adresu e-mail itd.;
- wszystkie wyświetlane przez aplet okna zawierają ostrzeżenie.

Wszystko to jest możliwe, ponieważ aplety są *interpretowane* przez wirtualną maszynę Javy, a nie wykonywane bezpośrednio przez procesor komputera użytkownika. Ponieważ interpreter sprawdza wszystkie krytyczne instrukcje i obszary kodu, niebezpieczny (lub błędnie napisany) aplet prawie na pewno nie będzie mógł zawiesić komputera, skasować ważnych fragmentów pamięci lub zmienić priorytety ustalone przez system operacyjny.

W niektórych sytuacjach te ograniczenia okazują się zbyt radykalne. Dla przykładu, w intranecie korporacji na pewno mogłyby pojawić się aplety korzystające z lokalnych plików. Aby ustanowić różne stopnie bezpieczeństwa w różnych sytuacjach, korzystaj z *apletów podpisanych*. Aplety podpisane są przesyłane wraz z certyfikatem identyfikującym osobę, która „podpisała" aplet. Techniki kryptograficzne gwarantują autentyczność certyfikatu. Jeżeli ufasz tej osobie, możesz nadać apletowi dodatkowe prawa (podpisywanie kodu omówimy w *Java 2. Techniki zaawansowane*).

Jeżeli ufasz osobie, która podpisała aplet, możesz poinformować przeglądarkę, aby nadała apletowi większe przywileje. Np. możesz dać większy stopień swobody apletom pochodzącym z intranetu Twojej firmy niż tym z *www.hacker.com*. Umożliwiający konfigurację model ochrony Javy pozwala na ustanowienie takiego poziomu bezpieczeństwa, jakiego potrzebujesz. Apletom, którym ufasz całkowicie, możesz nadać te same przywileje, co lokalnym aplikacjom. Programom innych firm, o których wiesz, że mogą zawierać pewne niewielkie błędy, możesz nadać trochę niższy stopień zaufania. Aplety nieznanego pochodzenia mogą zostać ograniczone do „piaskownicy".

Podsumujmy powyższe rozważania. Java posiada trzy mechanizmy zarządzające bezpieczeństwem systemu:

1. Kod programu jest interpretowany przez wirtualną maszynę Javy, a nie wykonywany bezpośrednio.

2. Menedżer ochrony sprawdza wszystkie podejrzane operacje należące do biblioteki wykonawczej Javy.

3. Aplety mogą zostać podpisane, by użytkownicy wiedzieli, skąd pochodzą.

Model ochrony technologii ActiveX firmy Microsoft polega wyłącznie na trzecim mechanizmie. Jeżeli chcesz uruchomić kontrolę ActiveX, musisz jej całkowicie zaufać. Ten model sprawdza się, jeżeli programy pochodzą od małej liczby zaufanych dostawców, ale całkowicie zawodzą w przypadku sieci WWW. Jeżeli używasz Internet Explorera, możesz sprawdzić, w jaki sposób działa ActiveX. Aby zainstalować Java Plug-In w Internet Explorerze, musisz zaakceptować certyfikat firmy Sun. Certyfikat ten informuje tylko, że kod programu pochodzi od firmy Sun. Nie mówi absolutnie nic o jakości kodu. Gdy już zakończysz instalację, działanie programu Java Plug-In nie będzie kontrolowane.

Okna dialogowe w apletach

Aplet jest wyświetlany wewnątrz strony internetowej, w ramce o rozmiarach określonych przez wartości `WIDTH` i `HEIGHT`, będące parametrami znacznika HTML. To poważne ograniczenie. Wielu programistów zastanawia się, czy mogliby zbudować okno dialogowe, aby lepiej wykorzystać dostępną przestrzeń. Rzeczywiście, jest możliwe utworzenie ramki okna dialogowego. Poniżej znajduje się prosty aplet zawierający jeden przycisk o etykiecie *Kalkulator*. Gdy klikniesz ten przycisk, kalkulator pojawi się w nowym oknie.

Łatwo jest utworzyć okno dialogowe. Po prostu skorzystaj z `JFrame`, ale nie pisz wywołania `setDefaultCloseOperation`.

```
ramka = new RamkaKalkulatora();
ramka.setTitle("Kalkulator");
ramka.setSize(200, 200);
```

Gdy użytkownik klika przycisk, zmienia się stan ramki — jeżeli wcześniej była wyłączona, teraz staje się widoczna, i odwrotnie. Gdy klikniesz przycisk kalkulatora, nad stroną internetową pojawi się okno dialogowe. Gdy klikniesz ponownie, kalkulator zniknie.

```
JButton przyciskKalk = new JButton("Kalkulator");
przyciskKalk.addActionListener(new
 ActionListener()
 {
  public void actionPerformed(ActionEvent zd)
  {
   if (ramka.isVisible()) ramka.setVisible(false);
   else ramka.show();
  }
 });
```

Jest jednak coś, o czym musisz wiedzieć, zanim umieścisz aplet na swojej stronie. Aby zobaczyć, jak aplet wygląda na ekranie użytkownika, uruchom go w przeglądarce internetowej, a nie w przeglądarce apletów. Do kalkulatora zostanie dołączone ostrzeżenie (rysunek 10.9).

Rysunek 10.9. *Okno dialogowe wewnątrz przeglądarki internetowej*

We wczesnych wersjach przeglądarek ostrzeżenie brzmiało bardzo poważnie: `Untrusted Java Applet Window` (`Niegodne zaufania okno apletu Javy`). Każda następna wersja SDK tonowała ostrzeżenie — `Unauthenticated Java Applet Window` (`Nieautoryzowane okno apletu Javy`) lub `Warning: Java Applet Window` (`Ostrzeżenie: Okno apletu Javy`). Teraz brzmi ono po prostu `Java Applet Window` (`Okno apletu Javy`).

Wiadomość ta jest elementem systemu bezpieczeństwa przeglądarek. Przeglądarka internetowa upewnia się w ten sposób, że aplet nie wyświetli okna dialogowego, które użytkownik mógłby pomylić ze standardową aplikacją. Może się zdarzyć, że stronę odwiedzi niedoświadczony użytkownik, automatycznie uruchamiając aplet, a następnie przez przypadek wpisze w wyświetlone okno dialogowe hasło lub numer karty kredytowej, które to dane aplet wyśle serwerowi.

Aby uniknąć niebezpieczeństwa zaistnienia takich sytuacji, wszystkie okna dialogowe uruchamiane przez aplet zawierają etykietę ostrzeżenia. Jeżeli Twoja przeglądarka obsługuje podpisane aplety, możesz skonfigurować ją tak, aby w przypadku programów, którym ufasz, ostrzeżenie nie było wyświetlane.

Listing 10.5 zawiera kod klasy `ApletDialoguKalkulatora`. Kod klasy `PanelKalkulatora` z rozdziału 9. nie uległ zmianie, został więc pominięty.

Listing 10.5. *ApletDialoguKalkulatora.java*

```
 1. import java.awt.*;
 2. import java.awt.event.*;
 3. import javax.swing.*;
 4.
 5. public class ApletDialoguKalkulatora extends JApplet
 6. {
 7.    public void init()
 8.    {
 9.       // zbuduj ramkę zawierającą panel kalkulatora
10.
11.       ramka = new JFrame();
12.       ramka.setTitle("Kalkulator");
13.       ramka.setSize(200, 200);
14.       ramka.getContentPane().add(new PanelKalkulatora());
15.
16.       // dołącz przycisk powodujący wyświetlenie kalkulatora
17.
18.       JButton przyciskKalk = new JButton("Kalkulator");
19.       getContentPane().add(przyciskKalk);
20.
21.       przyciskKalk.addActionListener(new
22.          ActionListener()
23.          {
24.             public void actionPerformed(ActionEvent zd)
25.             {
26.                if (ramka.isVisible()) ramka.setVisible(false);
27.                else ramka.show();
28.             }
29.          });
30.    }
31.
32.    private JFrame ramka;
33. }
```

Znaczniki HTML oraz atrybuty apletów

Skoncentrujemy się teraz na znaczniku APPLET, mimo iż został on zanegowany w najnowszych wersjach specyfikacji HTML Konsorcjum W3. Omówimy go, ponieważ ani przeglądarka apletów firmy Sun, ani konwerter HTML Java Plug-In nie rozpoznają nowego znacznika OBJECT.

W swojej podstawowej formie przykład zastosowania znacznika APPLET wygląda następująco:

```
<APPLET CODE="NieWitajSwiecieAplet.class" WIDTH=100 HEGIHT=100>
```

Jak już widziałeś, atrybut CODE podaje nazwę pliku klasy i musi zawierać rozszerzenie .class; atrybuty WIDTH i HEIGHT określają rozmiar okna, w którym aplet zostanie wyświetlony. Szerokość i wysokość jest mierzona w pikselach. Potrzebujesz również odpowiedniego znacznika </APPLET>, który oznacza koniec elementów wymaganych przez aplet. Tekst pomiędzy znacznikami <APPLET> i </APPLET> zostanie wyświetlony wyłącznie wtedy, jeśli wyświetlenie apletu okaże się niemożliwe. Znaczniki te są wymagane — jeżeli któregoś brakuje, przeglądarka nie załaduje apletu.

Wszystkie podane informacje zostaną wyświetlone na stronie HTML, której kod może wyglądać np. tak:

```
<HTML>
<HEAD>
<TITLE>NieWitajSwiecieAplet</TITLE>
</HEAD>
<BODY>
Następna linijka tekstu jest wyświetlana z pomocą Javy:
<APPLET CODE="NieWitajSwicieAplet.class" WIDTH=100 HEIGHT=100>
Gdyby Twoja przeglądarka obsługiwała Javę,
w tym miejscu pojawiłby się aplet.
</APPLET>
</BODY>
</HTML>
```

> Zgodnie ze specyfikacją HTML znaczniki i atrybuty HTML, takie jak APPLET, mogą być pisane zarówno wielkimi, jak i małymi literami. Wielkość liter ma znaczenie tylko przy identyfikacji nazwy klasy apletu. Wielkość liter może też mieć znaczenie w przypadku innych elementów, podawanych w cudzysłowach, taki jak nazwy plików JAR, ale pod warunkiem, że serwer rozróżnia wielkie i małe litery.

Atrybuty określające położenie apletu

Poniżej znajdziesz krótkie omówienie atrybutów, których możesz (lub musisz) używać w definiowaniu znacznika APPLET, aby określić położenie apletu. Dla osób znających HTML wiele z tych atrybutów będzie wyglądać znajomo, ponieważ są one podobne do atrybutów znacznika IMG, używanego do umieszczania obrazu na stronie internetowej.

- WIDTH, HEIGHT

  Te atrybuty są wymagane — określają wysokość i szerokość apletu, mierzone w pikselach. W przeglądarce apletów ich wartości oznaczają początkowy rozmiar apletu. W przeglądarce apletów możesz zmieniać rozmiar okna. W przeglądarce internetowej rozmiar apletu nie może zostać zmieniony. Będziesz więc musiał zgadywać, ile miejsca powinieneś przeznaczyć na aplet, aby był czytelny dla wszystkich użytkowników.

- ALIGN

  Ten atrybut określa wyrównanie apletu. Masz dwa wyjścia. Aplet może znajdować się w określonym miejscu, a tekst będzie go opływał, albo aplet może zostać *umieszczony* w linii tekstu, tak jakby był olbrzymią literą. Pierwsze dwie wartości (LEFT i RIGHT) powodują umieszczenie apletu w stałym miejscu. Pozostałe wartości sprawiają, że aplet znajdzie się we wnętrzu tekstu.

  Wartości, które może przyjmować atrybut, zostały wyszczególnione w tabeli 10.1.

Rysunek 10.10 prezentuje wszystkie opcje wyrównania apletu umieszczonego we wnętrzu tekstu. Pionowy pasek na początku każdej linii to obraz graficzny. Ponieważ obraz jest wyższy od tekstu, możesz zauważyć różnice pomiędzy górnym i dolnym końcem linii a górnym i dolnym końcem tekstu.

Tabela 10.1. *Atrybuty położenia apletu*

Atrybut	Działanie
LEFT	Umieszcza aplet przy lewej krawędzi strony. Znajdujący się w danym miejscu tekst zostanie przesunięty na prawo od apletu
RIGHT	Umieszcza aplet przy prawej krawędzi strony. Znajdujący się w danym miejscu tekst zostanie przesunięty na lewo od apletu
BOTTOM	Dolny koniec apletu zostanie zrównany z dolnym końcem *tekstu* w aktualnej linii
TOP	Górny koniec apletu zostanie zrównany z górnym końcem aktualnej linii
TEXTTOP	Górny koniec apletu zostanie zrównany z górnym końcem *tekstu* w aktualnej linii
MIDDLE	Środek apletu zostanie zrównany z linią bazową *tekstu* w aktualnej linii
ABSMIDDLE	Środek apletu zostanie zrównany ze środkiem aktualnej linii
BASELINE	Dolny koniec apletu zostanie zrównany z linią bazową *tekstu* w aktualnej linii
ABSBOTTOM	Dolny koniec apletu zostanie zrównany z dolnym końcem aktualnej linii
VSPACE, HSPACE	Te dodatkowe atrybuty określają liczbę pikseli wolnej przestrzeni, jaka powinna się znaleźć u góry i u dołu (VSPACE) oraz po lewej i po prawej stronie apletu (HSPACE)

Rysunek 10.10.
Wyrównanie apletu

Atrybuty dotyczące kodu apletu

Poniższe atrybuty apletu informują przeglądarkę, w jaki sposób powinna odnaleźć kod apletu.

- CODE

  Ten atrybut podaje nazwę pliku klasy apletu. Nazwa klasy jest wykorzystywana z uwzględnieniem atrybutu CODEBASE (zob. poniżej) albo z uwzględnieniem aktualnej strony.

Jeżeli używasz relatywnej ścieżki dostępu, musi ona wskazywać na pakiet pliku klasy. Dla przykładu, jeżeli klasa apletu znajduje się w pakiecie `com.mojafirma`, atrybut `CODE` powinien wyglądać następująco: `CODE="com/mojafirma/MojAplet.class"`. Dozwolona jest również składnia `CODE="com.mojafirma.MojAplet.class"`. Jednak w przypadku tego atrybutu nie możesz używać absolutnej ścieżki dostępu. Jeżeli plik klasy znajduje się w innym katalogu, użyj atrybutu `CODEBASE`.

Atrybut `CODE` określa jedynie nazwę pliku, który zawiera klasę Twojego apletu. Oczywiście, Twój aplet może korzystać z innych klas. W momencie, gdy przeglądarka załaduje główną klasę apletu, otrzyma informacje o innych wymaganych klasach i wczyta je.

W znaczniku apletu musi znaleźć się albo atrybut `CODE`, albo atrybut `OBJECT` (zob. poniżej).

- `CODEBASE`

  Ten opcjonalny atrybut informuje przeglądarkę, że pliki klas znajdują się wewnątrz podkatalogu określonego przez ten atrybut. Na przykład, jeżeli aplet o nazwie `ApletKalkulatora` znajduje się w katalogu *MojeAplety*, a katalog *MojeAplety* znajduje się *poniżej* lokacji strony internetowej, powinieneś napisać:

```
<APPLET CODE="ApletKalkulatora.class" CODEBASE="MojeAplety" WIDTH=100 HEIGHT=150>
```

  Innymi słowy, informujesz przeglądarkę, że pliki są rozmieszczone następująco:

```
Katalog/
 ApletKalkulatora.html
 MojeAplety/
  ApletKalkulatora.class
```

- `ARCHIVE`

  Ten opcjonalny atrybut zawiera plik lub listę plików archiwalnych zawierających klasy oraz inne zasoby apletu (przejdź do podrozdziału poświęconego plikom JAR, aby dowiedzieć się więcej o plikach archiwalnych Javy). Pliki te są pobierane z serwera, zanim aplet zostanie załadowany. Ten mechanizm znacząco przyspiesza proces ładowania apletu, ponieważ do ściągnięcia pliku JAR, zawierającego wiele mniejszych plików, wystarcza tylko jedno żądanie HTTP. Kolejne pliki JAR rozdzielasz przy użyciu przecinka. Dla przykładu:

```
<APPLET CODE="ApletKalkulatora.class" ARCHIVE=
 "KlasyKalkulatora.jar,corejava/KlasyCoreJava.jar"
 WIDTH=100 HEIGHT=150>
```

Powyższy atrybut ma jedno bardzo ważne zastosowanie. Jeżeli Twój aplet używa zestawu Swing, ale w żaden inny sposób nie korzysta z narzędzi Java 2, możesz go uruchomić na przeglądarkach zgodnych z Java 1.1 (takich jak Netscape 4 lub Internet Explorer), dostarczając plik JAR zawierający wszystkie klasy Swing. Możesz uzyskać odpowiednie archiwum, ściągając dodatek Swing dla Java 1.1 z *http://java.sun.com/products/jfc/#download-swing*. Następnie musisz spakować swój aplet w plik JAR i załadować go wraz z plikiem JAR zestawu Swing. Aby tego dokonać, dopisz do kodu strony parametr `ARCHIVE` (zob. poniżej).

```
<APPLET CODE="ApletKalkulatora.class"
 ARCHIVE="KlasyApletuKalkulatora.jar,swing.jar"
 WIDTH=100 HEIGHT=150>
```

Plik *swing.jar* ma rozmiar ok. jednego megabajta, co oznacza, że załadowanie go trochę potrwa. Lepiej więc, abyś nie korzystał z tego rozwiązania, jeżeli wiesz, że niektórzy użytkownicy będą łączyć się z Twoją stroną za pośrednictwem modemu. Oczywiście, Java Plug-In jest jeszcze większy, ale wystarczy ściągnąć go tylko raz. Plik JAR ładowany jest za każdym razem.

■ `OBJECT`

Innym sposobem określania pliku klasy apletu jest podanie nazwy pliku zawierającego serializowany obiekt apletu — jednakże niektóre przeglądarki nie obsługują tej opcji. Jeżeli chcesz z niej skorzystać, na pewno będziesz musiał zainstalować Java Plug-In (obiekt jest *serializowany*, gdy zapisujesz do pliku wartości wszystkich pól składowych tego obiektu; serializację omówimy w rozdziale 12.). Aby powrócić do pierwotnego stanu, obiekt jest deserializowany z pliku. Jeżeli używasz tego atrybutu zamiast metody `init`, wywoływana jest metoda `start` apletu. Przed serializowaniem obiektu apletu powinieneś wywołać metodę `stop`. Atrybut `OBJECT` jest użyteczny np. w tworzeniu strony, która będzie automatycznie przeładowywać aplety i wymuszać na nich powrót do tego samego stanu, w jakim znajdowały się, gdy przeglądarka została zamknięta. Jest to dość wyspecjalizowane narzędzie, zazwyczaj niewykorzystywane w projektach stron internetowych.

W znaczniku apletu musi znaleźć się albo atrybut `CODE`, albo atrybut `OBJECT`.

```
<APPLET OBJECT="ApletKalkulatora.ser" WIDTH=100 HEIGHT=150>
```

■ `NAME`

Programiści języków skryptowych zazwyczaj nadają apletom nazwy, aby móc manipulować nimi w skryptach. Zarówno Netscape Navigator, jak i Internet Explorer, pozwalają Ci wywoływać metody apletu za pomocą JavaScriptu. Nasza książka nie zajmuje się JavaScriptem, więc omówimy jedynie najważniejsze elementy tego języka skryptowego, za pomocą których możesz wywoływać kod Javy.

JavaScript jest językiem skryptowym wykorzystywanym przez strony internetowe. Język ten został opracowany przez firmę Netscape i początkowo nosił nazwę LiveScript. Nie ma on wiele wspólnego z Javą, może poza pewnymi podobieństwami w składni. Zmiana nazwy na JavaScript była posunięciem marketingowym. Podzbiór elementów tego języka (noszący chwytliwą nazwę ECMAScript) otrzymał standard jako ECMA-262. Ale, co nikogo już nie dziwi, Netscape i Microsoft używają w swoich przeglądarkach różnych, niezgodnych ze sobą rozszerzeń tego standardu. Aby dowiedzieć się więcej o JavaScript, zajrzyj do książki „JavaScript: The Definite Guide", autorstwa Davida Flanagana, wydanej przez O'Reilly & Associates.

Aby skorzystać z apletu w kodzie JavaScript, musisz najpierw nadać mu nazwę.

```
<APPLET CODE="ApletKalkulatora.class"
WIDTH=100 HEIGHT=150
NAME="kalk">
</APPLET>
```

Możesz teraz korzystać z niego, używając `document.applets.`*`nazwaapletu`*. Na przykład:

```
var apletKalk = document.applet.kalk;
```

Dzięki magii integracji Javy i JavaScript, zaprogramowanej zarówno w Netscape, jak i w Internet Explorerze, możesz wywoływać metody apletu:

```
apletKalk.clear();
```

(Nasz aplet kalkulatora nie posiada metody `clear`; chcieliśmy jedynie zaprezentować składnię.)

> Na stronie *http://www.javaworld.com/javatips/jw-javatip80.html* Francis Lu używa komunikacji JavaScript — Java, aby rozwiązać odwieczny problem: zmienić rozmiar apletu tak, aby nie był ograniczony przez podane wartości atrybutów `WIDTH` i `HEIGHT`. Jest to dobry przykład integracji języków Java i JavaScript, jak również ilustruje, ile kodu potrzeba, aby napisać skrypt działający pod różnymi przeglądarkami.

Atrybut `NAME` jest również konieczny, gdy chcesz, aby dwa aplety na tej samej stronie komunikowały się ze sobą bezpośrednio. Określasz nazwę każdej instancji apletu, następnie przekazujesz tę nazwę metodzie `getApplet` klasy `AppletContext`. Mechanizm ten, nazywany *komunikacją pomiędzy apletami* (ang. *inter-applet communication*), omówimy w dalszej części rozdziału.

Atrybuty dotyczące przeglądarek nie połączonych z Javą

Jeżeli strona internetowa zawierająca znacznik `APPLET` jest oglądana w przeglądarce nie obsługującej apletów, przeglądarka zignoruje znaczniki `APPLET` i `PARAM`. Tekst znajdujący się pomiędzy znacznikami `<APPLET>` i `</APPLET>` zostanie wyświetlony. Zwróć uwagę, że przeglądarki połączone z Javą nie wyświetlają tekstu znajdującego się pomiędzy tymi znacznikami. Dla tych biednych ludzi, którzy używają prehistorycznych przeglądarek, możesz w tym miejscu zamieścić tekst wyświetlany zamiast apletu. Na przykład:

```
<APPLET CODE="ApletKalkulatora.class" WIDTH=100 HEIGHT=150>
Gdyby Twoja przeglądarka obsługiwała Javę,
w tym miejscu pojawiłby się kalkulator.
</APPLET>
```

Oczywiście, większość obecnych przeglądarek obsługuje Javę, ale ta obsługa może czasami zostać wyłączona — być może przez użytkownika, a być może przez cierpiącego na paranoję administratora systemu. Aby wspomóc te nieszczęśliwe duszyczki, możesz wyświetlić wiadomość, korzystając z atrybutu `ALT`.

```
<APPLET CODE="ApletKalkulatora.class" WIDTH=100 HEIGHT=150
ALT="Gdyby Twoja przeglądarka obsługiwała Javę,
w tym miejscu pojawiłby się kalkulator.">
```

Znacznik OBJECT

Znacznik `OBJECT` jest częścią standardu HTML 4.0, a Konsorcjum W3 zaleca, aby programiści używali go zamiast znacznika `APPLET`. Znacznik `OBJECT` posiada 35 różnych atrybutów, z których większość (jak np. `ONKEYDOWN`) jest istotna wyłącznie dla ludzi programujących w Dynamic HTML. Atrybuty pozycjonowania, takie jak `ALIGN` i `HEIGHT`, działają dokładnie tak samo, jak w przypadku znacznika `APPLET`. Kluczowym dla apletów Javy atrybutem znacznika `OBJECT`

jest atrybut CLASSID. Atrybut ten określa lokację obiektu. Oczywiście, znaczniki OBJECT mogą ładować różne rodzaje obiektów, takie jak aplety Javy lub komponenty ActiveX, czyli np. sam Java Plug-In. W atrybucie CODETYPE opisujesz charakter obiektu. Dla przykładu, typem kodu apletów Javy jest application/java. Oto znacznik OBJECT wczytujący aplet Javy:

```
<OBJECT
 CODETYPE="application/java"
 CLASSID="java:ApletKalkulatora.class"
 WIDTH=100 HEIGHT=150>
```

Zauważ, że po atrybucie CLASSID może występować atrybut CODEBASE, który działa dokładnie tak, jak w przypadku znacznika APPLET.

W aktualnych wersjach przeglądarek Netscape Navigator i Internet Explorer możesz już używać znacznika OBJECT, ale przeglądarka apletów oraz konwerter HTML Java Plug-In nie rozpoznają tego znacznika.

Znaczniki Java Plug-In

Java Plug-In zostaje załadowany jako rozszerzenie Netscape lub kontrolka ActiveX, przy użyciu znaczników EMBED lub OBJECT. Dla przykładu, ekwiwalentem znacznika:

```
<APPLET
 CODE="ApletKalkulatora.class"
 CODEBASE="MojeAplety"
 WIDTH=100
 HEIGHT=150>
<PARAM NAME="Czcionka" VALUE="Helvetica">
</APPLET>
```

w przeglądarce Netscape Navigator jest:

```
<EMBED TYPE="application/x-java-applet;version=1.3"
 PLUGINSPAGE="http://java.sun.com/products/plugin/1.3
  /plugin-install.html"
 CODE="ApletKalkulatora.class"
 CODEBASE="MojeAplety"
 WIDTH=100
 HEIGHT=150>
<PARAM NAME="Czcionka" VALUE="Helvetica">
</EMBED>
```

Natomiast w Internet Explorerze:

```
<OBJECT CLASSID="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
 CODEBASE="http://java.sun.com/products/plugin/1.1
  /jinstall-11-win32.cab#Version=1,3,0,0"
 WIDTH=100
 HEIGHT=150>
<PARAM NAME="TYPE" VALUE="application/x-java-applet;
 version=1.3">
<PARAM NAME="CODE" VALUE="ApletKalkulatora.class">
<PARAM NAME="CODEBASE" VALUE="MojeAplety">
<PARAM NAME="Czcionka" VALUE="Helvetica">
</OBJECT>
```

Poniżej znajdują się detale konwertowania znaczników, jeżeli koniecznie chcesz dokonać konwersji własnoręcznie.

Konwersja znacznika APPLET na znacznik EMBED jest prosta — zmień APPLET na EMBED, a także dołącz atrybuty TYPE i PLUGINSPAGE.

Konwersja znacznika APPLET na znacznik OBJECT jest bardziej złożona. Musisz dopisać atrybuty CLASSID i CODEBASE oraz dołączyć znacznik PARAM o nazwie TYPE (CLASSID ma zawsze tę samą wartość; jest to globalny numer ID ActiveX, oznaczający Java Plug-In). Zachowaj wszystkie atrybuty, oprócz tych znajdujących się w tabeli 10.2, ponieważ muszą one zostać skonwertowane na znaczniki PARAM. Jeżeli może zaistnieć konflikt pomiędzy nimi a już wpisanymi znacznikami PARAM, w nazwach parametrów możesz użyć prefiksu JAVA_; na przykład:

```
<PARAM NAME="JAVA_CODE" VALUE="ApletKalkulatora.class">
```

Tabela 10.2. *Konwertowanie znacznika APPLET na znacznik OBJECT*

APPLET	OBJECT
ALT=...	brak
ARCHIVE=...	<PARAM NAME="ARCHIVE" VALUE=...>
CODE=...	<PARAM NAME="CODE" VALUE=...>
CODEBASE=...	<PARAM NAME="CODEBASE" VALUE=...>
OBJECT=...	<PARAM NAME="OBJECT" VALUE=...>

Jak widzisz, różnice pomiędzy tymi znacznikami sprowadzają się do kosmetyki. W praktyce najłatwiej jest używać konwertera HTML Java Plug-In albo innego skryptu, który automatycznie wygeneruje odpowiedni kod HTML.

Konwerter HTML Java Plug-In generuje również kod, który automatycznie wybiera znaczniki odpowiadające danej przeglądarce. W tym celu konwerter korzysta ze skryptów JavaScript lub używa niesamowicie zagmatwanej sekwencji znaczników, z których część jest ignorowana, zależnie od używanej przeglądarki. Aby dowiedzieć się więcej o tej koncepcji, przejrzyj dokumentację konwertera HTML lub stronę *http://java.sun.com/products/jfc/tsc/articles/plugin/index.html*.

Przekazywanie informacji apletom

Tak jak aplikacje mogą korzystać z informacji wpisanych w linii poleceń, tak aplety mogą używać parametrów zamieszczonych w pliku HTML. Możesz przekazać te informacje, używając znacznika PARAM, wraz ze zdefiniowanym przez Ciebie atrybutami. Dla przykładu załóżmy, że chcesz, aby strona HTML określała styl czcionki używanej przez aplet. W tym celu możesz skorzystać z następujących znaczników:

```
<APPLET CODE="ApletCzcionki.class" WIDTH=200 HEIGHT=200>
<PARAM NAME="czcionka" VALUE="Helvetica">
</APPLET>
```

Aplet pobiera wartość parametru, korzystając z metody `getParameter` klasy `Applet`, jak w poniższym przykładzie:

```
public class ApletCzcionki extends JApplet
{
 public void init()
 {
  String nazwaCzcionki = getParameter("czcionka");
  . . .
 }
 . . .
}
```

> Metodę `getParameter` możesz wywołać wyłącznie w metodzie `init`, nie w konstruktorze. Gdy uruchamiany jest konstruktor, parametry nie są jeszcze określone. Ponieważ układ graficzny większości apletów jest określany przez parametry, zalecamy, abyś nie pisał konstruktorów apletów. Cały kod inicjalizacyjny możesz po prostu umieścić w metodzie `init`.

Parametry zawsze są łańcuchami. Jeżeli parametr powinien być liczbą, musisz go skonwertować. Dokonujesz tego w standardowy sposób, używając odpowiedniej metody, np. `parseInt` klasy `Integer`.

Dla przykładu, gdybyś chciał dołączyć parametr rozmiaru czcionki, kod HTML mógłby wyglądać następująco:

```
<APPLET CODE="ApletCzcionki.class" WIDTH=200 HEIGHT=200>
<PARAM NAME="czcionka" VALUE=" ">
</APPLET> Helvetica">
<PARAM NAME="rozmiar" VALUE="24
```

Poniższy kod źródłowy ilustruje, w jaki sposób możesz wczytać parametr liczbowy.

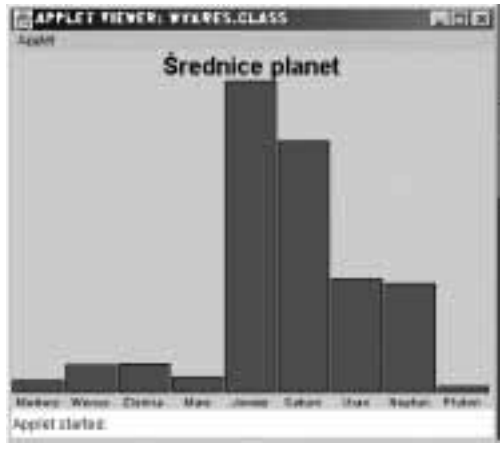
```
public class ApletCzcionki extends JApplet
{
 public void init()
 {
  String nazwaCzcionki = getParameter("czcionka");
  int rozmiarCzcionki = Integer.parseInt(getParameter("rozmiar"));
  . . .
 }
}
```

> Łańcuchy używane w definicji parametrów za pomocą znacznika `PARAM` muszą się dokładnie zgadzać z tymi w wywołaniach metody `getParameter`. W szczególności, w jednych i drugich istotna jest wielkość liter.

Aby dodatkowo upewnić się, czy parametry w Twoim kodzie zgadzają się ze sobą, powinieneś sprawdzać, czy nie został pominięty parametr `rozmiar`. Dokonujesz tego, sprawdzając, czy jego wartość wynosi `null`. Na przykład:

```
int rozmiarCzcionki;
String stringRozmiar = getParameter("rozmiar");
if (stringRozmiar == null) rozmiarCzcionki = 12;
else rozmiarCzcionki = Integer.parseInt(stringRozmiar);
```

Poniżej znajduje się użyteczny aplet korzystający z wielu parametrów. Aplet ten rysuje wykres kolumnowy przedstawiony na rysunku 10.11.



Rysunek 10.11.
Aplet „Wykres"

Aplet pobiera wysokości kolumn, korzystając ze znaczników PARAM w pliku HTML. Plik HTML tworzący rysunek 10.11 wygląda następująco:

```
<APPLET CODE="Wykres.class" WIDTH=400 HEIGHT=300>
<PARAM NAME="tytul" VALUE="Średnice planet">
<PARAM NAME="wartosci" VALUE="9">
<PARAM NAME="nazwa_1" VALUE="Merkury">
<PARAM NAME="nazwa_2" VALUE="Wenus">
<PARAM NAME="nazwa_3" VALUE="Ziemia">
<PARAM NAME="nazwa_4" VALUE="Mars">
<PARAM NAME="nazwa_5" VALUE="Jowisz">
<PARAM NAME="nazwa_6" VALUE="Saturn">
<PARAM NAME="nazwa_7" VALUE="Uran">
<PARAM NAME="nazwa_8" VALUE="Neptun">
<PARAM NAME="nazwa_9" VALUE="Pluton">
<PARAM NAME="wartosc_1" VALUE="3100">
<PARAM NAME="wartosc_2" VALUE="7500">
<PARAM NAME="wartosc_3" VALUE="8000">
<PARAM NAME="wartosc_4" VALUE="4200">
<PARAM NAME="wartosc_5" VALUE="88000">
<PARAM NAME="wartosc_6" VALUE="71000">
<PARAM NAME="wartosc_7" VALUE="32000">
<PARAM NAME="wartosc_8" VALUE="30600">
<PARAM NAME="wartosc_9" VALUE="1430">
</APPLET>
```

Oczywiście, mogłeś umieścić w aplecie tablicę łańcuchów oraz tablicę liczb, ale z użycia mechanizmu PARAM wynikają dwie istotne korzyści. Na swojej stronie internetowej możesz mieć wiele kopii tego samego apletu, ukazującego różne wykresy — po prostu umieszczasz na stronie dwa znaczniki APPLET, zawierające różne zbiory parametrów. Co więcej, możesz zmieniać dane, które chcesz umieścić na wykresie. Przyznajemy, że średnice planet nie zmienią się w najbliższym czasie, ale załóżmy na przykład, że Twoja strona zawiera wykres wyników sprzedaży w danym tygodniu. Aktualizacja strony internetowej jest prosta, ponieważ HTML to czysty tekst. Cotygodniowa edycja i rekompilacja pliku Javy jest już bardziej skomplikowana.

W rzeczywistości istnieją komercyjne programy JavaBeans tworzące bardziej wymyślne wykresy niż nasz aplet. Jeżeli kupisz jeden z nich, możesz umieścić go na stronie i dostarczać wymagane parametry, nie wiedząc nawet, w jaki sposób będzie on rysował wykres.

Listing 10.6 zawiera kod źródłowy naszego rysującego wykresy apletu. Zwróć uwagę, że metoda `init` wczytuje parametry, a metoda `paintComponent` rysuje wykres.

Listing 10.6. *Wykres.java*

```
 1. import java.awt.*;
 2. import java.awt.font.*;
 3. import java.awt.geom.*;
 4. import javax.swing.*;
 5.
 6. public class Wykres extends JApplet
 7. {
 8.    public void init()
 9.    {
10.       String w = getParameter("wartosci");
11.       if (w == null) return;
12.       int n = Integer.parseInt(w);
13.       double[] wartosci = new double[n];
14.       String[] nazwy = new String[n];
15.       int i;
16.       for (i = 0; i < n; i++)
17.       {
18.          wartosci[i] = Double.parseDouble
19.             (getParameter("wartosc_" + (i + 1)));
20.          nazwy[i] = getParameter("nazwa_" + (i + 1));
21.       }
22.
23.       Container powZawartosci = getContentPane();
24.       powZawartosci.add(new PanelWykresu(wartosci, nazwy,
25.          getParameter("tytul")));
26.    }
27. }
28.
29. /**
30.    Panel rysujący wykres.
31. */
32. class PanelWykresu extends JPanel
33. {
34.    /**
35.        tworzy obiekt typu PanelWykresu.
36.        @param w tablica wartości wykresu
37.        @param n tablica nazw wartości
38.        @param t tytuł wykresu
39.    */
40.    public PanelWykresu(double[] w, String[] n, String t)
41.    {
42.       nazwy = n;
43.       wartosci = w;
44.       tytul = t;
45.    }
46.
47.    public void paintComponent(Graphics g)
48.    {
```

```
49.      super.paintComponent(g);
50.      Graphics2D g2 = (Graphics2D)g;
51.
52.      // oblicz wartość minimalną i maksymalną
53.      if (wartosci == null) return;
54.      double warMinimalna = 0;
55.      double warMaksymalna = 0;
56.      for (int i = 0; i < wartosci.length; i++)
57.      {
58.         if (warMinimalna > wartosci[i]) warMinimalna = wartosci[i];
59.         if (warMaksymalna < wartosci[i]) warMaksymalna = wartosci[i];
60.      }
61.      if (warMaksymalna == warMinimalna) return;
62.
63.      int szerPanelu = getWidth();
64.      int wysPanelu = getHeight();
65.
66.      Font czcionkaTytulu = new Font("SansSerif", Font.BOLD, 20);
67.      Font czcionkaEtykiety = new Font("SansSerif", Font.PLAIN, 10);
68.
69.      // oblicz powierzchnię zajmowaną przez tytuł
70.      FontRenderContext kontekst = g2.getFontRenderContext();
71.      Rectangle2D krawedzieTytulu
72.         = czcionkaTytulu.getStringBounds(tytul, kontekst);
73.      double szerTytulu = krawedzieTytulu.getWidth();
74.      double gora = krawedzieTytulu.getHeight();
75.
76.      // narysuj tytuł
77.      double y = -krawedzieTytulu.getY(); // wzniesienie
78.      double x = (szerPanelu - szerTytulu) / 2;
79.      g2.setFont(czcionkaTytulu);
80.      g2.drawString(tytul, (float)x, (float)y);
81.
82.      // oblicz powierzchnię zajmowaną przez etykiety pasków
83.      LineMetrics metrykaEtykiety
84.         = czcionkaEtykiety.getLineMetrics("", kontekst);
85.      double dol = metrykaEtykiety.getHeight();
86.
87.      y = wysPanelu - metrykaEtykiety.getDescent();
88.      g2.setFont(czcionkaEtykiety);
89.
90.      // oblicz współczynnik skali oraz wysokość pasków
91.      double skala = (wysPanelu - gora - dol)
92.         / (warMaksymalna - warMinimalna);
93.      int szerKolumny = szerPanelu / wartosci.length;
94.
95.      // narysuj kolumny
96.      for (int i = 0; i < wartosci.length; i++)
97.      {
98.         // pobierz koordynaty dla prostokąta przedstawiającego kolumnę
99.         double x1 = i * szerKolumny + 1;
100.        double y1 = gora;
101.        double wysokosc = wartosci[i] * skala;
102.        if (wartosci[i] >= 0)
103.           y1 += (warMaksymalna - wartosci[i]) * skala;
104.        else
105.        {
106.           y1 += warMaksymalna * skala;
```

```
107.            wysokosc = -wysokosc;
108.         }
109.
110.         // wypełnij kolumnę kolorem i narysuj jej krawędź
111.         Rectangle2D prost = new Rectangle2D.Double(x1, y1,
112.            szerKolumny - 2, wysokosc);
113.         g2.setPaint(Color.red);
114.         g2.fill(prost);
115.         g2.setPaint(Color.black);
116.         g2.draw(prost);
117.
118.         // pod kolumną narysuj wyśrodkowaną etykietę
119.         Rectangle2D krawedzieEtykiety
120.            = czcionkaEtykiety.getStringBounds(nazwy[i], kontekst);
121.
122.         double szerEtykiety = krawedzieEtykiety.getWidth();
123.         x = i * szerKolumny + (szerKolumny - szerEtykiety) / 2;
124.         g2.drawString(nazwy[i], (float)x, (float)y);
125.      }
126.   }
127.
128.   private double[] wartosci;
129.   private String[] nazwy;
130.   private String tytul;
131. }
```

java.applet.Applet

- `public String getParameter(String nazwa)` pobiera parametr zdefiniowany przez dyrektywę `PARAM` znajdującą się na stronie, która ładuje dany aplet. Wielkość liter w łańcuchu ma znaczenie.
- `public String getAppletInfo()` jest to metoda, którą wielu autorów apletów przeładowuje, aby zwracała łańcuch zawierający informacje o autorze, wersji oraz prawach autorskich danego apletu. Powinieneś udostępnić te informacje, przeładowując powyższą metodę w swojej klasie apletu.
- `public String[][] getParameterInfo()` jest to metoda, którą wielu autorów apletów przeładowuje, aby zwracała tablicę opcji znacznika `PARAM` dostarczanych przez aplet. Każdy rząd zawiera trzy komórki: nazwę, typ oraz opis parametru. Oto przykład:

```
"fps", "1-10", "ramki na sekundę"
"powtorz", "boolean", "czy powtórzyć pętlę obrazów?"
"obrazy", "url", "katalog zawierający obrazy"
```

Multimedia

Aplety mogą obsługiwać zarówno pliki graficzne, jak i dźwiękowe. W chwili, gdy piszemy te słowa, pliki graficzne muszą mieć format GIF lub JPEG, dźwiękowe — AU, AIFF, WAV lub MIDI. Animowane pliki GIF są akceptowane i uruchamiane. Zazwyczaj pliki zawierające te informacje są określane przez adres URL, który musimy wcześniej pobrać.

Adresy URL

URL jest opisem zasobu znajdującego się w Internecie. Dla przykładu, `"http://java.sun.com/index.html"` informuje przeglądarkę, aby użyła protokołu przesyłania hipertekstu dla pliku *index.html* znajdującego się w witrynie *java.sun.com*.

Java posiada klasę `URL` opisującą adresy URL. Najprostszym sposobem utworzenia adresu URL jest przekazanie łańcucha jako parametru konstruktora `URL`:

```
URL u = new URL("http://java.sun.com/index.html");
```

Taki adres nazywamy *absolutnym* adresem URL, ponieważ określamy kompletną ścieżkę dostępu do zasobu. Innym użytecznym konstruktorem `URL` jest konstruktor adresu *relatywnego*.

```
URL dane = new URL(u, "dane/planety.dat");
```

Dzięki temu określamy położenie pliku *planety.dat* znajdującego się w podkatalogu *dane* adresu `u`.

Obydwa konstruktory upewniają się, czy podając URL, użyłeś prawidłowej składni. Jeżeli nie, wywołują `MalformedURLException` (wyjątek złego formatu URL). Jest to jeden z wyjątków, których kompilator nie pozwoli Ci zignorować. Odpowiedni kod wygląda następująco:

```
try
{
 String s = "http://java.sun.com/index.html";
 URL u = new URL(s);
 . . .
}
catch(MalformedURLException wyjatek)
{
 // kod obsługujący błąd
 wyjatek.printStackTrace();
}
```

Składnię obsługi wyjątków omówimy w rozdziale 12. Do tego czasu, jeżeli napotkasz kod podobny do tego z naszych przykładowych programów, po prostu przejdź do porządku dziennego nad słowami kluczowymi `try` i `catch`.

Typowym sposobem uzyskiwania adresu URL jest zapytanie apletu, skąd pochodzi, czyli:

- Jaki jest adres URL strony wywołującej aplet?
- Jaki jest adres URL samego apletu?

Aby uzyskać pierwszy z nich, skorzystaj z metody `getDocumentBase`; aby otrzymać drugi, skorzystaj z `getCodeBase`. Tych metod nie musisz umieszczać w bloku `try`.

> Za pośrednictwem apletów oraz Java Plug-In możesz uzyskać dostęp do bezpiecznych stron internetowych (*https*) — przeczytaj *http://java.sun.com/products/plugin/1.3/docs/https.html*. Aby tego dokonać, programy korzystają z mechanizmów SSL przeglądarki.

Tworzenie plików multimedialnych

Za pomocą metody `getImage` oraz `getAudioClip` można utworzyć obrazy oraz pliki audio. Na przykład:

```
Image kot = getImage(getDocumentBase(), "obrazy/kot.gif");
AudioClip miau = getAudioClip(getDocumentBase(), "audio/miau.au");
```

W powyższym przykładzie używamy metody `getDocumentBase()`, zwracającej URL, z którego aplet został ściągnięty. Drugi argument konstruktora `URL` określa lokalizację względem dokumentu bazowego, pod którą znajduje się obraz lub plik dźwiękowy (aplety nie muszą posługiwać się obiektami typu `Toolkit`, aby otrzymać plik graficzny).

> Pliki graficzne i dźwiękowe muszą znajdować się na tym samym serwerze, z którego pochodzi aplet. Ze względów bezpieczeństwa aplety nie mają dostępu do plików na innych serwerach („aplety mogą dzwonić wyłącznie do domu").

Kiedy już załadujesz obrazy i pliki audio, to jakie operacje możesz na nich przeprowadzać? W rozdziale 7. dowiedziałeś się, w jaki sposób można wyświetlać pojedynczy obraz. W *Java 2. Techniki zaawansowane*, w rozdziale dotyczącym wielowątkowości, znajdziesz informacje odnośnie wyświetlania animacji złożonej z wielu obrazów. Aby uruchomić plik audio, wywołujesz po prostu metodę `play`.

Możesz również wywołać metodę `play` bez wcześniejszego załadowania pliku audio.

```
play(getDocumentBase(), "audio/miau.au");
```

Jednakże aby wyświetlić obraz, musisz go najpierw załadować.

Aby przyspieszyć ładowanie, obiekty multimediów można przechowywać w plikach JAR (przejdź do następnego podrozdziału). Metody `getImage` oraz `getAudioClip/play` przeszukują automatycznie pliki JAR apletu. Jeżeli obraz lub plik dźwiękowy jest przechowywany w pliku JAR, zostanie natychmiast załadowany. W przeciwnym wypadku przeglądarka zażąda go od serwera.

java.net.URL

- `URL(String nazwa)` tworzy obiekt URL na podstawie łańcucha opisującego absolutny adres URL.
- `URL(String baza, String nazwa)` tworzy relatywny obiekt URL. Jeżeli łańcuch `nazwa` opisuje absolutny adres URL, łańcuch `baza` zostanie zignorowany. W przeciwnym wypadku łańcuch `nazwa` zostanie potraktowany jako podkatalog adresu URL opisywanego przez `baza`.

java.applet.Applet

- `URL getDocumentBase()` pobiera adres URL strony, na której znajduje się dany aplet.
- `URL getCodeBase()` pobiera adres URL, pod którym znajduje się sam aplet.

- `void play(URL url)`
- `void play(URL url, String nazwa)` pierwsza wersja metody uruchamia plik audio określony przez podany URL. Druga wersja używa łańcucha określającego relatywną ścieżkę dostępu względem adresu podanego w pierwszym argumencie. Jeżeli plik nie zostanie odnaleziony, nic się nie dzieje.
- `AudioClip getAudioClip(URL url)`
- `AudioClip getAudioClip(URL url, String nazwa)` pierwsza wersja metody pobiera plik audio znajdujący się pod podanym adresem. Druga wersja używa łańcucha określającego relatywną ścieżkę dostępu względem adresu podanego w pierwszym argumencie. Jeżeli plik nie zostanie odnaleziony, metoda zwraca `null`.
- `Image getImage(URL url)`
- `Image getImage(URL url, String nazwa)` zwraca obiekt typu Image, zawierający obraz znajdujący się pod podanym adresem. Jeżeli obraz nie istnieje, metoda natychmiast zwraca `null`. W przeciwnym wypadku uruchamiany jest osobny wątek pobierający obraz. Otrzymywanie plików zostało opisane w rozdziale 7.

Kontekst apletu

Aplet działa wewnątrz przeglądarki internetowej lub przeglądarki apletów. Aplet może zażądać od przeglądarki wykonania pewnych operacji, na przykład dostarczenia mu pliku audio, wyświetlenia krótkiej wiadomości w polu statusu lub wyświetlenia innej strony internetowej. Przeglądarka może wykonać te polecenia lub zignorować je. Dla przykładu, jeżeli aplet uruchamiany wewnątrz przeglądarki apletów zażąda od przeglądarki wyświetlenia strony internetowej, nic się nie stanie.

Komunikując się z przeglądarką, aplet wywołuje metodę `getAppletContext`. Metoda ta zwraca obiekt implementujący interfejs typu `AppletContext`. O konkretnej implementacji interfejsu `AppletContext` możesz myśleć jako o kablu łączącym aplet z przeglądarką. Oprócz `getAudioClip` i `getImage`, interfejs `AppletContext` zawiera kilka innych metod, które omówimy w poniższych podrozdziałach.

Komunikacja pomiędzy apletami

Strona internetowa może zawierać więcej niż jeden aplet. Jeżeli strona zawiera wiele apletów o tym samym atrybucie `CODEBASE`, mogą one komunikować się ze sobą. Oczywiście, jest to dość skomplikowane zagadnienie, którym rzadko będziesz się zajmował.

Jeżeli dla każdego apletu w pliku HTML podasz atrybut `NAME`, będziesz mógł używać metody `getApplet(String)` interfejsu `AppletContext`, aby pobrać referencję dowolnego apletu. Dla przykładu, jeżeli plik HTML zawiera znacznik

```
<APPLET CODE="Wykres.class" WIDTH=100 HEIGHT=100 NAME="Wykres1">
```

to wywołanie

```
Applet wykres1 = getAppletContext().getApplet("Wykres1");
```

zwróci referencję tego apletu. Co możesz zrobić z tą referencją? O ile w klasie `Wykres` umieściłeś metodę pozwalającą na pobranie nowych danych i ponowne narysowanie wykresu, możesz ją wywołać, dokonując odpowiedniego rzutowania.

```
((Wykres)wykres1).zmienDane(3, "Ziemia", 9000);
```

Możesz również pobrać listę apletów na stronie, nawet jeżeli nie posiadają one atrybutu `NAME`. Metoda `getApplets` zwraca tak zwany *obiekt enumeracji* (o enumeracji dowiesz się więcej z *Java 2. Techniki zaawansowane*). Oto pętla wyświetlająca nazwy klas wszystkich apletów znajdujących się na aktualnej stronie:

```
Enumeration e = getAppletContext().getApplets();
while (e.hasMoreElements())
{
 Object a = e.nextElement();
 System.out.println(a.getClass().getName());
}
```

Aplety nie mogą komunikować się z apletami pochodzącymi z innych stron internetowych.

Wyświetlanie elementów w przeglądarce

Z poziomu apletu masz dostęp do dwóch obszarów przeglądarki internetowej: pola statusu, oraz obszaru wyświetlania stron internetowych. Obydwa obszary używają metod klasy `AppletContext`.

Możesz wyświetlić łańcuch w polu statusu przy dolnej krawędzi przeglądarki, używając metody `showStatus`, np.:

```
showStatus("Ładowanie danych... proszę czekać");
```

> Z naszego doświadczenia wynika, że metoda `showStatus` nie jest zbyt użyteczna. Przeglądarka również używa pola statusu i dość często zdarza się, że nadpisuje Twoją drogocenną wiadomość czymś w stylu `"Ładowanie apletu"`. Korzystaj z pola statusu w przypadku mało znaczących wiadomości, jak np. `"Ładowanie danych... proszę czekać"`, ale nie w sytuacji, gdy użytkownik mógłby mieć problemy z powodu przeoczenia wiadomości.

Za pomocą metody `showDocument` możesz zażądać od przeglądarki, aby wyświetliła inną stronę internetową. Możesz tego dokonać na kilka sposobów. Najprostszym jest wywołanie `showDocument` z jednym argumentem — adresem URL, pod który chcesz się dostać.

```
URL u = new URL("http://java.sun.com/index.html");
getAppletContext().showDocument(u);
```

Problem polega na tym, że powyższe wywołanie otwiera nową stroną w tym samym oknie, w którym znajduje się strona aktualna, zatrzymując tym samym aplet. Aby powrócić do apletu, użytkownik musi kliknąć przycisk *Wstecz* w swojej przeglądarce.

Możesz poinformować przeglądarkę, aby wyświetliła aplet w innym oknie, podając w wywołaniu `showDocument` drugi parametr. Powinien to być łańcuch. Jeżeli będzie to specjalny łańcuch, `"_blank"`, przeglądarka, zamiast zmieniać zawartość aktualnego okna, otworzy nowe, zawierające podany dokument. Co ważniejsze, jeżeli skorzystasz z ramek HTML, będziesz mógł podzielić okno przeglądarki na kilka ramek, z których każda będzie miała

własną nazwę. Będziesz mógł umieścić aplet w jednej z ramek, a w drugiej wyświetlać przy jego użyciu różne dokumenty. W następnym podrozdziale zaprezentujemy przykład takiego rozwiązania.

Tabela 10.3 zawiera wszystkie możliwe argumenty metody `showDocument`.

Tabela 10.3. *Argumenty showDocument*

Drugi argument metody showDocument	Położenie
`"_self"` lub brak	Wyświetla dokument w aktualnej ramce
`"_parent"`	Wyświetla dokument w ramce rodzica
`"_top"`	Wyświetla dokument w ramce bazowej dla wszystkich innych
`"_blank"`	Wyświetla nowe, nienazwane okno najwyższego poziomu
Każdy inny łańcuch	Wyświetla dokument w ramce o podanej nazwie. Jeżeli ramka o tej nazwie nie istnieje, otwiera nowe okno i nadaje mu tę nazwę

`java.applet.Applet`

- `public AppletContext getAppletContext()` zwraca uchwyt środowiska przeglądarki internetowej. W przypadku większości przeglądarek możesz użyć tej informacji do kontrolowania przeglądarki, w której uruchamiany jest aplet.
- `void showStatus(String wiad)` wyświetla podany łańcuch w polu statusu przeglądarki.
- `AudioClip getAudioClip(URL url)` zwraca obiekt typu `AudioClip`, przechowujący plik dźwiękowy określony przez podany adres URL. Aby go uruchomić, skorzystaj z metody `play`.

`java.applet.AppletContext`

- `Enumeration getApplets()` zwraca enumerację (więcej informacji w *Java 2. Techniki zaawansowane*) wszystkich apletów w aktualnym kontekście, czyli na tej samej stronie internetowej.
- `Applet getApplet(String nazwa)` zwraca aplet aktualnego kontekstu o podanej nazwie, a jeżeli taki aplet nie istnieje, zwraca `null`. Przeszukiwana jest wyłącznie aktualna strona internetowa.
- `void showDocument(URL url)`
- `void showDocument(URL url, String cel)` wyświetla nowy dokument w podanej ramce przeglądarki. Pierwsza wersja metody powoduje zamianę aktualnej strony na nową. Druga wersja używa łańcucha identyfikującego docelową ramkę. Łańcuch `cel` może wyglądać następująco: `"_self"` (strona wyświetlana w aktualnej ramce, dokładnie tak samo jak w przypadku pierwszej wersji tej metody), `"_parent"` (strona wyświetlana w ramce rodzica), `"_top"` (strona wyświetlana w ramce najwyższego poziomu) oraz `"_blank"` (strona wyświetlana w nowym, nienazwanym oknie najwyższego poziomu). Łańcuch `cel` może również być nazwą docelowej ramki.

> Przeglądarka apletów firmy Sun nie wyświetla stron internetowych. Polecenia `showDocument` są przez nią ignorowane.

Aplet „Spis adresów”

Poniższy aplet korzysta z ramek standardu HTML 3.2 i nowszych. Podzieliliśmy ekran na dwie pionowe ramki. Ramka po lewej zawiera aplet Javy wyświetlający listę adresów internetowych. Gdy wybierzesz któryś z nich, aplet wyświetli w ramce po prawej odpowiednią stronę (patrz rysunek 10.12).

Rysunek 10.12. *Aplet „Spis adresów”*

Listing 10.7 zawiera plik HTML definiujący ramki.

Listing 10.7. *SpisAdresow.html*

```
 1. <HTML>
 2. <HEAD>
 3. <TITLE>Aplet "Spis adresów"</TITLE>
 4. </HEAD>
 5. <FRAMESET COLS="320,*">
 6. <FRAME NAME="lewa" SRC="Lewa.html"
 7.    MARGINHEIGHT=2 MARGINWIDTH=2
 8.    SCROLLING = "no" NORESIZE>
 9. <FRAME NAME="prawa" SRC="Prawa.html"
10.    MARGINHEIGHT=2 MARGINWIDTH=2
11.    SCROLLING = "yes" NORESIZE>
12. </FRAMESET>
13. </HTML>
```

Nie będziemy omawiać składni tego pliku. Istotne jest to, że każda z ramek posiada dwa elementy: nazwę (podaną przez atrybut `NAME`) oraz adres URL (podany przez atrybut `SRC`). Nie mogliśmy wymyślić żadnych dobrych nazw dla ramek, więc nazwaliśmy je po prostu `"lewa"` i `"prawa"`.

Ramka po lewej (patrz listing 10.8) wczytuje plik o nazwie *Lewa.html*, który ładuje aplet. Plik *Lewa.html* określa po prostu dane apletu oraz listę adresów. Możesz edytować ten plik, zmieniając listę adresów i dostosowując ją do swojej strony internetowej.

Listing 10.8. *Lewa.html (przed skonwertowaniem go przy użyciu konwertera HTML)*

```
 1. <HTML>
 2. <TITLE>Aplet "Spis adresów"</TITLE>
 3. <BODY>
 4. Kliknij jeden z poniższych przełączników.
 5. Odpowiadająca mu strona internetowa
 6. zostanie wyświetlona w ramce po prawej.
 7. <P>
 8. <APPLET CODE="SpisAdresow.class" WIDTH=290 HEIGHT=300>
 9. <PARAM NAME=link_1 VALUE="http://java.sun.com">
10. <PARAM NAME=link_2 VALUE="http://www.gamelan.com">
11. <PARAM NAME=link_3 VALUE="http://www.zhongwen.com">
12. <PARAM NAME=link_4 VALUE="http://www.horstmann.com">
13. <PARAM NAME=link_5 VALUE="http://www.prenhall.com">
14. <PARAM NAME=link_6 VALUE="http://usps.com">
15. <PARAM NAME=link_7 VALUE="http://www.netscape.com">
16. </APPLET>
17. </BODY>
18. </HTML>
```

Ramka po prawej (patrz listing 10.9) wczytuje prosty plik, który nazwaliśmy *Prawy.html* (przeglądarka Netscape nie pozwala na istnienie ramek nie posiadających plików, musieliśmy więc dostarczyć plik wczytywany na samym początku).

Listing 10.9. *Prawa.html*

```
1. <HTML>
2. <TITLE>
3. Tu będą wyświetlane strony internetowe.
4. </TITLE>
5. <BODY>
6. Kliknij jeden z przełączników po lewej.
7. Tutaj zostanie wyświetlona odpowiednia strona internetowa.
8. </BODY>
9. </HTML>
```

Kod apletu „Spis adresów”, zawarty w listingu 10.10, jest dość prosty. Wczytuje wartości parametrów `link_1`, `link_2` itd., i zmienia każdą z nich w przełącznik. Gdy wybierzesz jeden z przełączników, metoda `showDocument` wyświetli odpowiednią stronię internetową w ramce po prawej.

Listing 10.10. *SpisAdresow.java*

```
 1. import java.awt.*;
 2. import java.awt.event.*;
 3. import java.applet.*;
 4. import java.util.*;
 5. import java.net.*;
 6. import javax.swing.*;
 7.
 8. public class SpisAdresow extends JApplet
 9. {
10.    public void init()
11.    {
12.       Box pudelko = Box.createVerticalBox();
13.       ButtonGroup grupa = new ButtonGroup();
```

```
14.
15.        int i = 1;
16.        String łańcuchURL;
17.
18.        // wczytaj wszystkie parametry link_n
19.        while ((łańcuchURL = getParameter("link_" + i)) != null)
20.        {
21.
22.           try
23.           {
24.              final URL url = new URL(łańcuchURL);
25.
26.              // dla każdego linku utwórz przełącznik
27.              JRadioButton przycisk = new JRadioButton(łańcuchURL);
28.              pudelko.add(przycisk);
29.              grupa.add(przycisk);
30.
31.              // wybór przełącznika powoduje wyświetlenie adresu URL
32.              // w ramce "prawa"
33.              przycisk.addActionListener(new
34.                 ActionListener()
35.                 {
36.                    public void actionPerformed(ActionEvent zdarzenie)
37.                    {
38.                       AppletContext kontekst = getAppletContext();
39.                       kontekst.showDocument(url, "prawa");
40.                    }
41.                 });
42.           }
43.           catch(MalformedURLException wyjatek)
44.           {
45.              wyjatek.printStackTrace();
46.           }
47.
48.           i++;
49.        }
50.
51.        Container powZawartosci = getContentPane();
52.        powZawartosci.add(pudelko);
53.     }
54. }
```

To aplet. Nie, to aplikacja. To jedno i to samo!

Kilka lat temu telewizyjny program rozrywkowy „Saturday Night Live" wyemitował reklamę przedstawiającą małżeństwo sprzeczające się o białą, żelatynową substancję. Mąż mówił: „To przybranie do deserów". Żona mówiła: „To wosk do podłogi". A spiker odpowiadał triumfalnym głosem: „To jedno i to samo!".

Czytając ten podrozdział, nauczysz się pisać w Javie program, który będzie *równocześnie* apletem i aplikacją. Oznacza to, że będziesz mógł go załadować przy użyciu przeglądarki apletów lub przeglądarki internetowej, albo też możesz go uruchomić z linii poleceń za pomocą interpretera Javy. Trudno powiedzieć, jak często będzie Ci potrzebne takie rozwiązanie — pomyśleliśmy jednak, że jest całkiem interesujące i że na pewno warto je poznać.

Zrzuty ekranów przedstawione na rysunkach 10.13 i 10.14 przedstawiają *ten sam* program, wywołany z linii poleceń jako aplikacja oraz oglądany w przeglądarce apletów jako aplet.

Rysunek 10.13. *Kalkulator jako aplikacja*

Rysunek 10.14. *Kalkulator jako aplet*

Zobaczmy, jak tego dokonano. Każdy plik klasy posiada dokładnie jedną klasę publiczną. Aby przeglądarka apletów mogła ją uruchomić jako aplet, klasa ta musi rozszerzać klasę `Applet`. Aby Java mogła ją uruchomić jako aplikację, musi posiadać statyczną metodę `main`. A zatem mamy co następuje:

```
class MojApletAplikacja extends JApplet
{
 public void init() { . . . }
 . . .
 static public void main(String[] args) { . . . }
}
```

Co możemy umieścić w metodzie `main`? Zazwyczaj tworzymy obiekt jakiejś klasy i wywołujemy dla niego metodę `show`. Jednak w tym wypadku nie jest to takie proste. Nie możesz wyświetlić surowego apletu. Aplet musi znaleźć się wewnątrz ramki. A gdy już znajdzie się wewnątrz ramki, musi zostać wywołana metoda `init`.

Aby zbudować ramkę, tworzymy klasę `RamkaApletu`, jak poniżej:

```
public class RamkaApletu extends JFrame
{
 public RamkaApletu(Applet aApplet)
 {
  aplet = aApplet;
  Container powZawartosci = getContentPane();
  powZawartosci.add(aplet);
  . . .
 }
 . . .
}
```

Konstruktor ramki umieszcza aplet (pochodzący od klasy `Component`) wewnątrz nowej ramki.

W metodzie `main` naszego apletu (aplikacji) tworzymy nową ramkę powyższego typu.

```
class MojApletAplikacja extends JApplet
{
 public void init() { . . . }
 . . .
 public static void main(String args[])
 {
  RamkaApletu ramka
   = new RamkaApletu(new MojApletAplikacja());
  ramka.setTitle("MojApletAplikacja");
  ramka.setSize(SZEROKOSC, WYSOKOSC);
  ramka.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  ramka.show();
 }
}
```

Istnieje jeden problem. Jeżeli taki program zostanie uruchomiony przez interpreter Javy, a nie przez przeglądarkę apletów, i nastąpi wywołanie `getAppletContext`, metoda ta zwróci `null`, ponieważ aplet nie znajduje się wewnątrz przeglądarki. To z kolei spowoduje błąd wykonania programu za każdym razem, gdy interpreter spróbuje wykonać kod podobny do następującego:

```
getAppletContext().showStatus(wiadomosc);
```

O ile nie chcemy konstruować całej przeglądarki, musimy dostarczyć przynajmniej podstawowe elementy umożliwiające wykonanie takich wywołań. Powyższe wywołanie nie wyświetli żadnej wiadomości, ale przynajmniej nie spowoduje błędu programu. Okazuje się, że wszystko, co musimy zrobić, to zaimplementować dwa interfejsy: `AppletStub` i `AppletContext`.

Działanie kontekstów apletu już widziałeś. Są one odpowiedzialne za dostarczanie plików graficznych i dźwiękowych oraz za wyświetlanie stron internetowych. Mogą również grzecznie odmówić wykonania polecenia — i właśnie tak będzie postępował nasz kontekst apletu. Głównym zadaniem interfejsu `AppletStub` jest właśnie odnalezienie kontekstu apletu. Każdy aplet posiada procedurę pośredniczącą (ang. *applet stub*; jest ona określana przez metodę `setStub` klasy `Applet`).

W naszym przypadku klasa `RamkaApletu` będzie implementować zarówno `AppletStub`, jak i `AppletContext`. Dostarczamy najmniejszą możliwą ilość kodu, jaka pozwala zaimplementować te dwa interfejsy.

```
public class RamkaApletu extends JFrame
 implements AppletStub, AppletContext
{
 . . .
 // metody AppletStub
 public boolean isActive() { return true; }
 public URL getDocumentBase() { return null; }
 public URL getCodeBase() { return null; }
 public String getParameter(String nazwa) { return ""; }
 public AppletContext getAppletContext() { return this; }
 public void appletResize(int szerokosc, int wysokosc) {}
```

```
 // metody AppletContext
 public AudioClip getAudioClip(URL url){ return null; }
 public Image getImage(URL url){ return null; }
 public Applet getApplet(String nazwa){ return null; }
 public Enumeration getApplets(){ return null; }
 public void showDocument(URL url) {}
 public void showDocument(URL url, String cel) {}
 public void showStatus(String status) {}
}
```

Gdy skompilujesz ten plik, otrzymasz ostrzeżenie, że *java.awt.Window* również zawiera metodę o nazwie `isActive`, posiadającą widoczność pakietu. Ponieważ nasza klasa nie znajduje się w tym samym pakiecie co klasa `Window`, nie może przeładować metody `Window.isActive`. Nie przeszkadza nam ta sytuacja — chcemy utworzyć nową metodę `isActive` dla interfejsu `AppletStub`. Interesujący jest fakt, że dołączenie do podklasy nowej metody, o tej samej sygnaturze, co jedna z metod nadklasy, jest całkowicie legalne. Jeżeli w odniesieniu do obiektu, wewnątrz pakietu `java.awt`, będziemy używać referencji Window, wywołana zostanie metoda `Window.isActive`. Ale jeżeli skorzystamy z referencji `RamkaApletu` lub `AppletStub`, wywołana zostanie metoda `RamkaApletu.isActive`.

Następnie konstruktor ramki wywołuje dla apletu metodę `setStub`, aby samemu stać się procedurą pośredniczącą apletu.

```
 public RamkaApletu(Applet aAplet)
{
 aplet = aAplet;
 Container powZawartosci = getContentPane();
 powZawartosci.add(aplet);
 aplet.setStub(this);
}
```

Możliwa jest jeszcze jedna sztuczka. Załóżmy, że chcemy korzystać z kalkulatora równocześnie jako apletu i jako aplikacji. Zamiast przenosić metody klasy `ApletKalkulatora` do klasy `KalkulatorApletAplikacja`, skorzystamy z dziedziczenia. Oto kod klasy, która tego dokonuje:

```
public class KalkulatorApletAplikacja extends ApletKalkulatora
{
 public static void main(String args[])
 {
  RamkaApletu ramka
   = new RamkaApletu(new ApletKalkulatora());
  . . .
 }
}
```

Możesz zastosować to rozwiązanie w każdym aplecie, nie tylko w aplecie kalkulatora. Wszystko, co musisz zrobić, to wywieść klasę `MojApletAplikacja` z klasy apletu i w metodzie `main` przekazać klasie `RamkaApletu` obiekt `new MojAplet()`. W wyniku tego powstanie klasa będąca równocześnie apletem i aplikacją.

Dla zabawy skorzystaliśmy ze sztuczki, o której wspomnieliśmy już wcześniej — polega ona na dołączeniu znacznika `APPLET` jako komentarza w pliku źródłowym. Dzięki temu możesz wywołać przeglądarkę apletów dla pliku źródłowego (!), omijając dodatkowy plik HTML.

Listingi 10.11 i 10.12 zawierają kod źródłowy. Aby skompilować program, musisz skopiować plik *ApletKalkulatora.java* do tego samego katalogu. Spróbuj uruchomić zarówno aplet, jak i aplikację:

```
appletviewer KalkulatorApletAplikacja.java
java KalkulatorApletAplikacja
```

Listing 10.11. *RamkaApletu.java*

```
 1. import java.awt.*;
 2. import java.awt.event.*;
 3. import java.aplet.*;
 4. import java.net.*;
 5. import java.util.*;
 6. import javax.swing.*;
 7.
 8. public class RamkaApletu extends JFrame
 9.    implements AppletStub, AppletContext
10. {
11.    public RamkaApletu(Applet aAplet)
12.    {
13.       aplet = aAplet;
14.       Container powZawartosci = getContentPane();
15.       powZawartosci.add(aplet);
16.       aplet.setStub(this);
17.    }
18.
19.    public void show()
20.    {
21.       aplet.init();
22.       super.show();
23.       aplet.start();
24.    }
25.
26.    // metody AppletStub
27.    public boolean isActive() { return true; }
28.    public URL getDocumentBase() { return null; }
29.    public URL getCodeBase() { return null; }
30.    public String getParameter(String nazwa) { return ""; }
31.    public AppletContext getAppletContext() { return this; }
32.    public void appletResize(int szerokosc, int wysokosc) {}
33.
34.    // metody AppletContext
35.    public AudioClip getAudioClip(URL url) { return null; }
36.    public Image getImage(URL url) { return null; }
37.    public Applet getApplet(String nazwa) { return null; }
38.    public Enumeration getApplets() { return null; }
39.    public void showDocument(URL url) {}
40.    public void showDocument(URL url, String cel) {}
41.    public void showStatus(String status) {}
42.
43.    private Applet aplet;
44. }
```

Listing 10.12. *KalkulatorApletAplikacja.java*

```
 1. /*
 2.   Jeżeli wywołasz appletviewer KalkulatorApletAplikacja.java,
 3.   przeglądarka apletów wczyta poniższe znaczniki. (!)
 4.   Osobny plik HTML nie jest potrzebny.
 5.   <APPLET CODE="KalkulatorApletAplikacja.class"
 6.      WIDTH=200 HEIGHT=200>
 7.   </APPLET>
 8. */
 9.
10. import javax.swing.*;
11.
12. public class KalkulatorApletAplikacja
13.    extends ApletKalkulatora
14. // To aplet. To aplikacja. To jedno i to samo!
15. {
16.    public static void main(String[] args)
17.    {
18.       RamkaApletu ramka
19.          = new RamkaApletu(new ApletKalkulatora());
20.       ramka.setTitle("KalkulatorApletAplikacja");
21.       ramka.setSize(SZEROKOSC, WYSOKOSC);
22.       ramka.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23.       ramka.show();
24.    }
25.
26.    public static final int SZEROKOSC = 200;
27.    public static final int WYSOKOSC = 200;
28. }
```

Pliki JAR

Aplet kalkulatora z tego rozdziału korzysta z czterech klas: `ApletKalkulatora`, `PanelKalkulatora` oraz dwóch klas wewnętrznych. Wiesz, że znacznik apletu wskazuje na plik klasy, zawierający klasę pochodzącą od `JApplet`:

```
<APPLET CODE="ApletKalkulatora.class" WIDTH=100 HEIGHT=150>
```

Gdy przeglądarka wczytuje tę linię, łączy się z serwerem i pobiera plik *ApletKalkulatora.class*. Następnie *mechanizm ładowania klas* interpretera Javy, wbudowany w przeglądarkę, ładuje z tego pliku klasę `ApletKalkulatora`. Podczas tego procesu mechanizm ładowania klas musi *rozwiązać* inne klasy, z których korzysta klasa główna. Gdy tego dokona, wie już, jakich jeszcze klas potrzebuje, aby uruchomić aplet. Dlatego też przeglądarka musi ponownie połączyć się z serwerem. Większość apletów korzysta z wielu klas, a przeglądarka internetowa musi osobno ściągać każdy z plików. Ściąganie apletu przy użyciu powolnego połączenia może zająć wiele minut.

> Ważne, abyś zapamiętał, że czas ładowania jest w dużej mierze niezależny od wielkości klas — ich pliki są dość małe. Ważniejszym powodem jest fakt, że ustanowienie każdego połączenia z serwerem zabiera dłuższą chwilę.

Java obsługuje ulepszoną metodę ładowania plików klas, która pozwala umieścić wszystkie wymagane pliki klas w jednym pliku. Plik ten może zostać ściągnięty za pomocą *jednego* żądania HTTP. Pliki archiwizujące pliki klas Javy są nazywane plikami archiwalnymi Javy (ang. *Java Archive files*, *JAR*). Pliki JAR mogą zawierać zarówno pliki klas, jak i inne typy plików, jak np. obrazy lub pliki dźwiękowe. Pliki JAR są kompresowane przy użyciu popularnego formatu kompresji ZIP, co dodatkowo redukuje czas ładowania.

Aby zbudować plik JAR, używasz narzędzia o nazwie `jar` (jeśli wybrałeś domyślną instalację, znajdziesz je w katalogu *jdk/bin*). Najbardziej typowe polecenie tworzące plik JAR ma następującą składnię:

```
jar cvf NazwaPlikuJAR Plik1 Plik2 . . .
```

Dla przykładu:

```
jar cvf KlasyKalkulatora.jar *.java icon.gif
```

Ogólnie rzecz biorąc, polecenie `jar` ma następujący format:

```
jar opcje Plik1 Plik2 . . .
```

Tabela 10.4 zawiera listę opcji programu `jar`. Są one dość podobne do opcji polecenia `tar` w systemie Unix.

Tabela 10.4. *Opcje programu jar*

Opcja	Opis
`c`	Tworzy nowe, puste archiwum i dodaje do niego pliki. Jeżeli którakolwiek z podanych nazw plików oznacza katalog, program `jar` rekursywnie przetwarza jego zawartość
`t`	Wyświetla tabelę zawartości
`u`	Aktualizuje istniejący plik JAR
`x`	Wydobywa pliki. Jeżeli podałeś jeden lub więcej nazw plików, tylko te pliki zostaną wydobyte z archiwum. W przeciwnym wypadku wydobywane są wszystkie pliki
`f`	Informuje program, że drugim parametrem jest nazwa pliku JAR. Jeżeli ten parametr nie pojawi się, program `jar` wypisze wynik w standardowym wyjściu (tworząc plik JAR) lub wczyta dane ze standardowego wejścia (wydobywając pliki lub wyświetlając zawartość pliku JAR)
`v`	Wyświetla opis działania programu
`m`	Dodaje do pliku JAR *manifest*. Manifest to opis zawartości oraz pochodzenia archiwum. Każde archiwum ma domyślny manifest, ale możesz dostarczyć własny, jeżeli chcesz poświadczyć autentyczność archiwum. Omówimy to zagadnienie w rozdziale poświęconym bezpieczeństwu, w *Java 2. Techniki zaawansowane*
`0`	Zachowuje dane bez użycia kompresji ZIP
`M`	Nie tworzy pliku manifestu dla zawartości archiwum
`i`	Tworzy indeks (poniżej znajdziesz więcej informacji na ten temat)
`C`	Tymczasowo zmienia katalog. Na przykład `jar cvf NazwaPlikuJAR.jar -C klasy *.class` przechodzi do katalogu *klasy* i z niego pobiera pliki klas

Gdy już masz plik JAR, musisz poinformować o nim przeglądarkę, używając znacznika APPLET, tak jak w poniższym przykładzie.

```
<APPLET CODE="ApletKalkulatora.class"
 ARCHIVE="KlasyKalkulatora.jar"
 WIDTH=100 HEIGHT=150>
```

Zwróć uwagę, że atrybut CODE nadal musi być obecny. Atrybut CODE podaje przeglądarce nazwę apletu. ARCHIVE określa po prostu plik, w którym znajduje się klasa apletu oraz inne pliki. Kiedykolwiek potrzebna jest klasa, obraz lub plik dźwiękowy, przeglądarka jako pierwsze przeszukuje pliki JAR znajdujące się na liście atrybutu ARCHIVE. Dopiero gdy okaże się, że wymagany plik nie znajduje się w archiwum, przeglądarka spróbuje ściągnąć go z serwera.

Manifest

Pliki JAR nie są wykorzystywane wyłącznie przez aplety. Możesz umieszczać w nich także aplikacje komponenty programów (czasami nazywane „JavaBeans" — przejdź do rozdziału 8. książki *Java 2. Techniki zaawansowane*) oraz biblioteki. Na przykład, biblioteka wykonawcza SDK znajduje się w olbrzymim pliku *rt.jar*.

Plik JAR to po prostu plik ZIP zawierający klasy, inne pliki wymagane przez program (takie jak ikony) oraz plik *manifestu* opisujący właściwości archiwum.

Plik manifestu nosi nazwę *MANIFEST.MF* i znajduje się wewnątrz pliku JAR, w specjalnym podkatalogu *META-INF*. Minimalna dozwolona zawartość manifestu to po prostu napis:

```
Manifest-Version: 1.0
```

Złożone manifesty mogą posiadać o wiele więcej komórek. Komórki manifestu są pogrupowane w sekcje. Pierwszą sekcją manifestu jest tzw. *sekcja główna*. Odnosi się ona do całego pliku JAR. Kolejne komórki mogą opisywać właściwości elementów, takich jak poszczególne pliki, pakiety lub adresy URL. Komórki w sekcji muszą rozpoczynać się od komórki Name. Poszczególne sekcje są rozdzielone pustymi liniami, np.:

```
Manifest-Version: 1.0
linijki opisujące archiwum

Name: Cokolwiek.class
linijki opisujące ten plik

Name: fu/pasek/
linijki opisujące ten pakiet
```

Aby edytować manifest, umieść linijki, które chcesz dodać do manifestu, w pliku tekstowym. Następnie uruchom co następuje:

```
jar cfm NazwaPlikuJAR NazwaPlikuManifestu . . .
```

Dla przykładu, aby utworzyć nowy plik JAR zawierający manifest, uruchom:

```
jar cfm MojeArchiwum.jar manifest.mf com/mojafirma/mojpkt/*.class
```

Aby do manifestu istniejącego pliku JAR dołączyć nowe elementy, umieść je w pliku tekstowym i użyj polecenia:

```
jar cfm MojeArchiwum.jar manifest-dodatki.mf
```

Aby dowiedzieć się więcej o mechanizmie JAR oraz formacie manifestu, zajrzyj na stronę *http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html*.

> Jeżeli Twój aplet jest duży, istnieje szansa, że użytkownicy nie będą używać wszystkich jego elementów. Aby zredukować czas ładowania, możesz podzielić kod apletu na wiele plików JAR, a do głównego pliku JAR dodać *indeks*. Mechanizm ładowania klas będzie dzięki temu wiedział, który z plików JAR zawiera dany pakiet lub zasób. Aby wygenerować indeks, musisz w manifeście głównego pliku JAR określić atrybut `Class-Path`. Następnie uruchom
>
> ```
> jar -i MojGlownyAplet.jar
> ```
>
> Powyższe polecenie dołącza do katalogu *META-INF* plik *INDEX.LIST*.

Przechowywanie plików JAR

Do przechowywania plików JAR przeglądarki domyślnie używają swojej pamięci podręcznej. Tak więc, jeżeli powrócisz do strony zawierającej aplet, a przeglądarka wciąż będzie przechowywać odpowiedni plik JAR w pamięci podręcznej, nie zostanie on ponownie ściągnięty z serwera. To dobre rozwiązanie, ale pamięć podręczna przeglądarki nie jest tak pomocna, jakby chcieli tego programiści apletów. Na przykład, jeżeli raz w miesiącu odwiedzasz stronę zawierającą aplet rozliczający wydatki, prawdopodobnie za każdym razem będziesz musiał pobierać go od nowa.

Java Plug-In obsługuje mechanizm czyniący aplety bardziej „wytrzymałymi". Jeżeli chcesz, aby aplet pozostał po stronie użytkownika przez dłuższy okres czasu, skorzystaj ze słów kluczowych `CACHE_OPTION`, `CACHE_ARCHIVE` i `CACHE_VERSION`.

Powyższe słowa kluczowe muszą stać się parametrami lub atrybutami — zależnie od tego, czy używasz znacznika `OBJECT`, czy `EMBED`, w następujący sposób:

```
<OBJECT ...>
<PARAM NAME="ARCHIVE" VALUE="...">
...
<PARAM NAME="CACHE_OPTION" VALUE="...">
<PARAM NAME="CACHE_ARCHIVE" VALUE="...">
<PARAM NAME="CACHE_VERSION" VALUE="...">
</OBJECT>
```

lub

```
<EMBED ...
 ARCHIVE="..."
 CACHE_OPTION="..."
 CACHE_ARCHIVE="..."
 CACHE_VERSION="..."
 ...
>
```

Słowo kluczowe `CACHE_OPTION` może przyjmować trzy wartości:

- `No` — nie przechowuj apletu w pamięci podręcznej.
- `Browser` — pozwól przechowywać aplet przeglądarce (domyślnie).
- `Plugin` — pozwól przechowywać aplet Java Plug-In.

> Plik JAR powinien znajdować się na liście atrybutu `ARCHIVE` lub `CACHE_ARCHIVE`, ale nie na obydwu równocześnie.

Para `CACHE_VERSION`/wartość jest opcjonalna. Wartością jest lista numerów wersji, odpowiadających plikom JAR na liście `CACHE_ARCHIVE`, które reprezentują wymagane wersje plików JAR. Jeżeli na komputerze klienta znajdują się odpowiednie wersje, to znaczy, że nie trzeba ich pobierać. Jeżeli numery wersji nie zostały określone, w porównaniu używane są daty plików JAR. Numerów wersji musisz używać, gdy przechowujesz pliki JAR otrzymane za pomocą SSL, ponieważ w takich sytuacjach data ostatniej modyfikacji jest niedostępna dla Java Plug-In.

Aby dowiedzieć się więcej o przechowywaniu apletów, zajrzyj na *http://java.sun.com/products/plugin/1.3/docs/appletcaching.html*.

Autonomiczne pliki JAR

Do tej pory omawialiśmy pakowanie apletów w pliki JAR. W przypadku apletów format JAR jest wykorzystywany głównie do zredukowania liczby plików, jakie przeglądarka musi ściągnąć, używając osobnych żądań HTTP. Jednak pliki JAR są przydatne również w dystrybucji aplikacji, gdyż zmniejszają liczbę plików zaśmiecających komputery użytkowników. Po prostu umieść wszystkie pliki aplikacji w pliku JAR i dołącz do manifestu komórkę opisującą *główną klasę* Twojego programu — klasę, którą normalnie podałbyś, wywołując interpreter `java`.

Utwórz plik, powiedzmy, *klasaglowna.mf*, zawierający następującą linijkę:

```
Main-Class: com/mojpakiet/RamkaGlowna
```

Nie dodawaj rozszerzenia `.class` do nazwy głównej klasy. Następnie uruchom program `jar`:

```
jar cvfm MojProgram.jar klasaglowna.mf dołączane pliki
```

Użytkownicy mogą teraz uruchamiać Twój program w następujący sposób:

```
java -jar MojProgram.jar
```

Zasoby

Klasy, używane zarówno w apletach, jak i w aplikacjach, często korzystają z przyporządkowanych sobie plików danych, takich jak:

- pliki graficzne i dźwiękowe,

- pliki tekstowe, zawierające łańcuchy wiadomości i etykiety przycisków,
- pliki binarne, opisujące np. mapę.

W Javie taki przyporządkowany plik nazywany jest *zasobem*.

> W systemie Windows termin „zasób" ma bardziej konkretne znaczenie. Zasoby Windows również zawierają obrazy, etykiety przycisków itd., ale są one połączone z plikiem wykonywalnym i można mieć do nich dostęp poprzez standardowy interfejs programowania. W przeciwieństwie do tego, zasoby Javy są przechowywane w osobnych plikach, nie jako część plików klas. Każda klasa z osobna określa sposób dostępu i interpretację danych zasobu.

Dla przykładu, spójrzmy na klasę `PanelInformacji` wyświetlającą wiadomość, np. taką jak na rysunku 10.15.

Rysunek 10.15. *Wyświetlanie zasobu pochodzącego z pliku JAR*

Oczywiście, tytuł książki i rok uzyskania praw autorskich zmienią się wraz z następną edycją książki. Aby można było łatwo wyśledzić tę zmianę, umieścimy tekst wewnątrz pliku, zamiast zapisać go w programie jako łańcuch.

Ale gdzie powinieneś umieścić plik *informacja.txt*? Oczywiście, najbardziej wygodnym rozwiązaniem byłoby umieszczenie go w tym samym miejscu, co inne pliki, czyli np. w pliku JAR.

Mechanizm ładowania klas wie, jak odszukać pliki klas, o ile ich lokalizacja jest zapisana w ścieżce dostępu, albo też znajdują się w archiwum lub na serwerze. Mechanizm zasobów daje Ci tę samą wygodę w przypadku plików, które nie są plikami klas. Oto, co musisz zrobić:

1. Pobierz obiekt typu `Class` klasy, która korzysta z danego zasobu, np. `PanelInformacji.class`.
2. Wywołaj `getResource(nazwapliku)`, aby otrzymać lokalizację zasobu w formie adresu URL.
3. Jeżeli dany zasób jest obrazem lub plikiem dźwiękowym, wczytaj go bezpośrednio, za pomocą metody `getImage` lub `getAudioClip`.
4. W przeciwnym wypadku wywołaj metodę `openStream` dla obiektu URL, aby załadować dane tego pliku (aby dowiedzieć się więcej o strumieniach danych, przejdź do rozdziału 12.).

Sztuczka polega na tym, że mechanizm ładowania klas pamięta, gdzie znajduje się klasa, i może spróbować odszukać zasób w tym samym katalogu.

Dla przykładu, aby zbudować ikonę zawierającą obraz z pliku *informacja.gif*, napisz następujące linijki:

```
URL url = PanelInformacji.class.getResource("informacja.gif");
ImageIcon ikona = new ImageIcon(url);
```

Oznacza to „odszukaj plik *informacja.gif* w tym samym miejscu, w którym znalazłeś `PanelInformacji.class`".

Aby wczytać plik *informacja.txt* używasz analogicznych instrukcji:

```
URL url = PanelInformacji.class.getResource("informacja.txt");
InputStream we = url.openStream();
```

Ponieważ powyższa kombinacja wywołań jest dość popularna, napisano dla niej skrót — metoda `getResourceAsStream` zwraca `InputStream`, a nie URL.

```
InputStream we = PanelInformacji.class.getResourceAsStream("informacja.txt");
```

Aby wczytać dane ze strumienia, musisz wiedzieć, w jaki sposób obsługiwać proces wejścia (w rozdziale 12. znajdziesz szczegóły). W naszym przykładowym programie sczytujemy ze strumienia po jednej linii naraz, używając poniższych instrukcji:

```
InputStream we = PanelInformacji.class.
 getResourceAsStream("informacja.txt");
BufferedReader odczyt = new BufferedReader(new
 InputStreamReader(we));
String linia;
while ((linia = odczyt.readLine()) != null)
 obszarTekstowy.append(linia + "\n");
```

Na płycie CD znajdziesz plik JAR zawierający wszystkie pliki klas tego przykładu oraz pliki zasobów *informacja.gif* i *informacja.txt*. Dzięki temu możemy zademonstrować, w jaki sposób aplet lokalizuje plik zasobu w tym samym miejscu, co plik klasy, czyli wewnątrz pliku JAR.

> Jak przekonałeś się w poprzednim rozdziale, w pliku JAR możesz umieścić obrazy oraz pliki dźwiękowe i operować na nich za pomocą metod `getImage` i `getAudioClip` — metody te automatycznie przeszukują pliki JAR. Ale jeżeli chcesz załadować z archiwum JAR inne pliki, musisz skorzystać z metody `getResourceAsStream`.

Zamiast umieszczać plik zasobu w tym samym katalogu, co plik klasy, możesz umieścić go w podkatalogu. Możesz skorzystać z hierarchicznej nazwy zasobu, takiej jak

```
dane/tekst/informacja.txt
```

Jest to relatywna nazwa pliku, interpretowana względem pakietu klasy, która ładuje zasób. Zwróć uwagę, że zawsze musisz używać separatora /, niezależnie od tego, jakiego separatora używa do rozdzielania katalogów Twój system operacyjny. Na przykład w Windows mechanizm ładowania zasobów automatycznie zamieni separatory / na \.

Nazwa zasobu, rozpoczynająca się od /, nazywana jest absolutną nazwą zasobu. W takim wypadku zasób jest lokalizowany w taki sam sposób, w jaki lokalizowana byłaby klasa znajdująca się wewnątrz pakietu. Dla przykładu, zasób

```
/corejava/tytul.txt
```

znajduje się w katalogu *corejava* (który może być wpisany w ścieżkę dostępu, znajdować się wewnątrz pliku JAR lub na serwerze).

Mechanizm ładowania zasobów pozwala jedynie na automatyzację procesu ładowania plików. Nie istnieją standardowe metody interpretujące zawartość pliku zasobu. Każdy aplet sam musi implementować interpretację zawartości swoich zasobów.

Innym popularnym zastosowaniem zasobów jest internacjonalizacja apletów i aplikacji. Zależne od używanego języka łańcuchy, takie jak wiadomości i etykiety interfejsu użytkownika, są przechowywane w plikach zasobów — jeden plik przypada na jeden język. *Internacjonalizacja API*, którą omawiamy w rozdziale 10. książki *Java 2. Techniki zaawansowane*, obsługuje standardowe metody organizowania i dostępu do tych plików.

Listing 10.13 zawiera kod HTML testujący ładowanie zasobów; listing 10.14 zawiera kod Javy.

Listing 10.13. *TestZasobow.html*

```
1. <APPLET CODE="TestZasobow.class"
2.    WIDTH=300 HEIGHT=200
3.    ARCHIVE="TestZasobow.jar">
4. </APPLET>
```

Listing 10.14. *TestZasobow.java*

```
 1. import java.awt.*;
 2. import java.awt.event.*;
 3. import java.io.*;
 4. import java.net.*;
 5. import javax.swing.*;
 6.
 7. public class TestZasobow extends JApplet
 8. {
 9.    public void init()
10.    {
11.       Container powZawartosci = getContentPane();
12.       powZawartosci.add(new PanelInformacji());
13.    }
14. }
15.
16. /**
17.    Panel zawierający obszar tekstowy i przycisk Informacja. Naciśnięcie
18.    przycisku powoduje wypełnienie obszaru tekstowego tekstem pochodzącym z zasobu.
19. */
20. class PanelInformacji extends JTextArea
21. {
22.    public PanelInformacji()
23.    {
24.       setLayout(new BorderLayout());
25.
26.       // dołącz obszar tekstowy
```

```
27.       obszarTekstowy = new JTextArea();
28.       add(new JScrollPane(obszarTekstowy), BorderLayout.CENTER);
29.
30.       // dołącz przycisk Informacja
31.       URL urlInformacji = PanelInformacji.class.getResource("informacja.gif");
32.       JButton przyciskInformacji = new JButton("Informacja",
33.          new ImageIcon(urlInformacji));
34.       przyciskInformacji.addActionListener(new Poinformowanie());
35.       add(przyciskInformacji, BorderLayout.SOUTH);
36.    }
37.
38.    private JTextArea obszarTekstowy;
39.
40.    private class Poinformowanie implements ActionListener
41.    {
42.       public void actionPerformed(ActionEvent zdarzenie)
43.       {
44.          try
45.          {
46.             // wczytaj tekst z zasobu w obszar tekstowy
47.             InputStream we = PanelInformacji.class.
48.                getResourceAsStream("informacja.txt");
49.             BufferedReader odczyt = new BufferedReader(new
50.                InputStreamReader(we));
51.             String linia;
52.             while ((linia = odczyt.readLine()) != null)
53.                obszarTekstowy.append(linia + "\n");
54.          }
55.          catch(IOException wyjatek)
56.          {
57.             wyjatek.printStackTrace();
58.          }
59.       }
60.    }
61. }
```

`java.lang.Class`

- `URL getResource(String nazwa)`
- `InputStream getResourceAsStream(String nazwa)` **odszukuje zasób w tym samym katalogu, w którym znajduje się aktualna klasa, a następnie zwraca adres URL lub strumień wejścia, z których możesz korzystać podczas ładowania zasobu. Obydwie metody zwracają** `null`**, jeżeli zasób nie został odnaleziony, tak więc nie zwracają wyjątku błędu wejścia-wyjścia.**

  Parametry: `Nazwa` **nazwa zasobu.**

Pakiety opcjonalne

Jak widziałeś, pliki JAR przydają się do archiwizowania zarówno apletów, jak i aplikacji. Często są również wykorzystywane do archiwizacji bibliotek kodu źródłowego. Możesz dołączyć do pliku JAR zbiór klas i udostępnić te klasy, dodając ten plik do ścieżki dostępu.

W przypadku często wykorzystywanych bibliotek kodu źródłowego możesz ominąć ten drugi krok, zamieniając plik JAR w *pakiet opcjonalny*.

> W początkowych wersjach J2SE pakiety opcjonalne były nazywane *rozszerzeniami*. Termin „pakiet" jest trochę mylący — pakiet opcjonalny może zawierać klasy pochodzące z różnych pakietów języka programowania Java.

Sekcja główna manifestu pakietu opcjonalnego opisuje zawartość archiwum. Oto przykład:

```
Extension-Name: com.mojafirma.mojerozszerzenie
Specification-Vendor: Moja Firma SA
Specification=Version: 1.0
Implementation-Vendor-Id: com.mojafirma
Implementation-Vendor: Moja Firma SA
Implementation-Version: 1.0.3
```

Nazwa rozszerzenia jest dowolna. Tak jak w przypadku pakietów języka Java, możesz zapewnić unikalność rozszerzenia, używając odwróconej nazwy domeny.

Program wymagający rozszerzeń opisuje je w sekcji głównej swojego manifestu:

```
Extension-List: mojeroz inneroz
mojeroz-Extension-Name: com.mojafirma.mojerozszerzenie
mojeroz-Specification-Version: 1.0
mojeroz-Implementation-Version: 1.0.1
mojeroz-Implementation-Vendor-Id: com.mojafirma
inneroz-Extension-Name: com.hal.util
inneroz-Specification-Version: 2.0
```

Ta konkretna aplikacja potrzebuje dwóch pakietów opcjonalnych, którym nadaliśmy *aliasy* `mojeroz` i `inneroz`. Linijka *alias*`-Extension-Name` podaje nazwę aktualnego rozszerzenia. Aplikacja informuje w ten sposób, że potrzebuje pakietu `mojeroz`, zgodnego ze specyfikacją 1.0 lub późniejszą i implementacją 1.0.1 lub późniejszą, oraz nalega, aby wymagany pakiet został zaimplementowany przez określonego dostawcę. W przypadku drugiego rozszerzenia program wymaga jedynie, aby było ono zgodne ze specyfikacją 2.0 lub nowszą. Implementacja i dostawca nie grają roli.

Gdy program zostanie uruchomiony, będzie musiał znaleźć swoje pakiety opcjonalne. Pakiet opcjonalny JAR może po prostu zostać umieszczony w katalogu *jre/lib/ext*. Program może również podać adres URL, spod którego pakiet opcjonalny zostanie pobrany, jak np.

```
alias-Implementation-URL: http://www.mojafirma.com/packages/mojerozszerzenie.jar
```

Aby dowiedzieć się więcej o pakietach opcjonalnych, zajrzyj na *http://java.sun.com/j2se/1.3/docs/guide/extensions/index.html*.

Pieczętowanie

W rozdziale 4. wspomnieliśmy, że możesz *zapieczętować* pakiet języka Java, aby upewnić się, że nie zostaną do niego dołączone żadne nowe klasy. Pieczętowanie zabezpiecza elementy o widoczności pakietu. Aby to osiągnąć, umieszczasz wszystkie klasy pakietu w pliku JAR.

Jednak domyślnie pakiety w plikach JAR nie są pieczętowane. Możesz zmienić to domyślne zachowanie, umieszczając linijkę

```
Sealed:true
```

w sekcji głównej manifestu archiwum. Możesz również określić, czy poszczególny pakiet ma zostać zapieczętowany, dodając do pliku JAR osobną sekcję, taką jak:

```
Name: com/mojafirma/mojpakiet/
Sealed: true

Name: com/hal/util/
Sealed: false
```

Aby zapieczętować pakiet, utwórz plik tekstowy zawierający instrukcje manifestu. Następnie uruchom program JAR:

```
jar cvfm MojPakiet.jar pieczec.mf dołączane pliki
```

W ten sposób kończymy omawianie apletów. W następnym rozdziale dowiesz się, jak używać mechanizmu wyjątków, aby informować swoje programy, co mają robić, jeśli w czasie wykonywania wystąpią problemy. Damy Ci również kilka wskazówek i omówimy techniki testowania i debugowania programów pomagające zmniejszyć liczbę sytuacji, w których Twoje programy nie będą działały prawidłowo.